Big Data Analytics in Structural Health Monitoring

By

Guowei Cai

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Civil Engineering

September 30, 2017

Nashville, Tennessee

Approved:

Sankaran Mahadevan, Ph.D.

Douglas Adams, Ph.D.

P. K. Basu, Ph.D.

Daniel Fabbri, Ph.D.

To my dear family

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest and sincere gratitude to my advisor, Prof. Sankaran Mahadevan, who provided guidance, knowledge and encouragement throughout my Ph.D. study at Vanderbilt University. He is a knowledgeable, kind and patient mentor.

I am also extremely grateful to my committee members: Prof. Douglas Adams, Prof. P. K. Basu, and Prof. Daniel Fabbri, for their insightful comments and suggestions. I would like to thank U.S. Department of Energy (DOE), for supporting the Light Water Reactor Sustainability (LWRS) Program. Dr. Vivek Agarwal and Dr. Bruce Hallbert from Idaho National Laboratory (INL) are very helpful. Also, Dr. Hai Huang from INL is a great mentor and friend to me, from whom I learnt a lot in my summer internship in 2015. Dr. Liping Wang and Dr. Arun Subramaniyan from GE Global Research Center were great mentors for me in summer internships in 2013 and 2014.

I am also grateful to work with so many brilliant people at Vanderbilt University. . In particular, I am very lucky to have Dr. Zhen, who gave me detailed and even hands-on help. I also thank my teammates from Prof. Mahadevan's group, including Dr. Chenzhao Li, Dr. You Ling, Dr. Chen Liang, Dr. Saideep Nannapaneni, Dr. Erin DeCarlo, Ghina Absi, Xiaoge Zhang, Nath Paromita, Kyle Neal, Eric Vanderhorn, Yanqing Bao, Dan Ao and Sarah Miele, for all the teamwork in coding, brainstorming in the whiteboard, and presentations in our group meeting. Also, I would like to thank Dr. Xiang Zhang, my best friend in Vanderbilt University for his support in friendship. I would like to thank all friends I met during my internships in INL and GE, who make that time precious and memorable.

Finally, I would like express my gratitude to my parents and brother, for their endless love and support as well as consistent encouragement in the tough time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

During the span of a structure's service life, conditions such as wear, overload, environmental degradation, and natural disasters may accelerate the degradation of the material and the structure. Structural health monitoring (SHM) is a vital tool to ensure that the structure is reliable within the design life, and also to potentially extend the service life beyond the designed life (Naus, 2009). SHM techniques can be either data-driven or model-based. In both cases, the data is often obtained using non-destructive evaluation (NDE) techniques, which can be divided into active and passive techniques. Examples of active NDE techniques are electromagnetic testing (ET) (Nagy, 2016) and ultrasonic guided wave testing (UGWT) (Yan et al., 2010). Examples of passive NDE techniques are acoustic emission (Nair and Cai, 2010), digital image correlation (DIC) (Roux et al., 2009), fiber-optic sensing (FOS) (Lopez-Higuera et al., 2011). Some other NDE techniques can be used in either active or passive modes, such as infrared thermography (IR) (Bagavathiappan et al., 2013). From the data type point of view, these monitoring techniques acquire either wave signals (ET, UGWT, AE), or images (DIC, IR). Data acquisition and analysis are crucial elements in structural health monitoring.

Structural health monitoring (SHM) aims to track the health state of a mechanical system, detect and diagnose any damage, and perform prognosis of future states (Balageas et al., 2006). Uncertainty occurs in all stages of SHM. In addition, due to modern advancements in sensor technology and increased capabilities for data collection and storage, the amount of acquired data is growing, which gradually increases the demands on data acquisition and analysis techniques. For example, 26 sensor arrays were used on the Vincent Thomas Bridge (VTB) in San Pedro, California generate 3 terabytes (TB) per year (Kallinikidou et al., 2013); in the health monitoring of wind turbine blades, over 300 GB of acoustic emission data were sampled during 6 months (Anastasopoulos et al., 2012); 7 GB of data were sampled per day in the Confederation Bridge Monitoring Project in Canada (Desjardins et al., 2006); and over 20 GB of data were obtained during automated railway inspection in the city of Brockton, MA (Zhang et al., 2014). All these applications call for the introduction of big data analytics into structural health monitoring. Mahadevan et al. (2014) pointed out the need for big data analytics as one of the four elements in an effective prognostics and health management framework for concrete structures. The big data issue mainly affects two elements in structural health monitoring: data acquisition and data analytics. For data acquisition, data synchronization is a critical problem to solve, especially in a wireless sensor network. Several researchers such as Araujo et al. (2012), Gandhi et al. (2007), and Yu (2012) have studied this problem.

Structural health monitoring involves several activities, namely, diagnosis with available data, design of experiments to facilitate effective diagnosis, and prognosis of future state given the inference on the current state. Although big data techniques are growing in number, effective big data analytics techniques in support of the above activities are yet to be developed.

For big data analytics in SHM diagnosis, Farrah et al. (2015) proposed an approach to analyze large scale wireless sensor network data. In this research, MapReduce was used to create the data tables and Hadoop was adopted to parallelize the detection method. Similar research has been done by parallelizing the time series analyses in Hadoop (Yu & Lin, 2015), and parallelizing the neural networks (NN)-based inference via MapReduce (Tran, 2015) in order to accomplish structural damage detection. However, big data analytics in the context of a Bayesian approach to SHM has not been reported. Therefore in this dissertation, the MapReduce technique will be investigated to parallelize particle filtering (PF) (Chatzi & Smyth, 2013), an effective Bayesian updating algorithm used in damage diagnosis and prognosis.

Big data analytics in damage prognosis is another challenge for SHM, and a few attempts to apply MapReduce framework for this purpose have been reported. The application of Hadoop in real-time bridge health monitoring was discussed by Roshandeh et al. (2014), who proposed a layered big data and a real-time decision-making framework for bridge data management as well as health monitoring. However, only a rough procedure was presented, and no uncertainties were incorporated. Similarly a framework for flood prediction has been studied by Kezia & Mary (2016). Challenges for reliability analysis in the context of big data were discussed by Meeker and Hong (2013); some applications were reviewed where field reliability data were used. This paper also explored opportunities to use modern reliability data in order to develop stronger statistical methods to operate and predict the performance of systems in the field. However, the focus was mainly on cost-effective usage of System Operation/Environmental (or SOE) data.

The above review shows that research has not yet been reported towards risk prognosis of existing structures, in the context of big data. Risk estimation requires the quantification of uncertainty arising from multiple sources – sensors, data analytics, and system models. Therefore this dissertation utilizes big data techniques to analyze voluminous SHM data for damage diagnosis, and to quantify the uncertainty in diagnosis and prognosis. Prognosis is realized using a damage growth model coupled with FEA, and remaining useful life (RUL) is predicted.

Field data is sometimes available in SHM, which can be used to update the model parameters for system identification (Park et al., 2006). Traditionally, data at only a few locations are used in system identification. Compressive sensing is used to minimize the number of points at which the field is measured (Di Ianni et al., 2015). This approach loses significant amount of information and reduces the accuracy of diagnosis. Ideally, the use of full field data is preferable, however, due to the expensiveness of computation, this hasn't been implemented and applied in SHM. This dissertation will explore this potential of efficient usage of high volume field data for diagnosis.

Uncertainty quantification methods require repeated evaluation of numerical models, which is often computationally expensive. One approach to overcome this challenge is to replace the original physics-based model with an inexpensive, efficient surrogate model. There are different surrogate modeling techniques, which can be divided into two types: response surrogate and distribution surrogate. A response surrogate aims to provide the output value for a given set of inputs as opposed to a distribution surrogate, which provides a distribution output for a given set of inputs. In other words, the distribution surrogate is constructed in the probability space whereas the response surrogate is constructed in the variable space. In this dissertation, one response surrogate model (Gaussian process surrogate model) and one distribution surrogate model (Gaussian mixture model) are used to illustrate the proposed methods.

1.2 Research Objectives

The first objective investigates techniques to perform diagnosis with large volume field data. Image processing techniques (such as uniform filtering and Sobel filtering) are used to analyze infrared thermal images, from which damage inside the structure can be detected. To handle the costly computation, big data techniques are employed to parallelize the computation. The methodology is illustrated through the detection of damage in a concrete slab, based on actual experimental data with induced damage.

The second objective investigates techniques to parallelize structural diagnosis and prognosis with uncertainty quantification. Both forward and inverse problems in uncertainty quantification are investigated with this efficient computational approach. We use Bayesian methods for the inverse problem of diagnosis, and parallelize sampling techniques such as Markov chain Monte Carlo simulation and particle filter. To predict damage growth and the structure's remaining useful life (forward problem), Monte Carlo simulation is used to propagate the uncertainties (both aleatory and epistemic) to the future state. The big data technique MapReduce is applied to drive the parallelization of multiple FEA runs, thus greatly saving the computational cost. The proposed techniques are illustrated for the efficient diagnosis and prognosis of alkali-silica reaction in a concrete structure.

The third objective investigates big data analytics for high-dimensional model parameter calibration, in order to facilitate accurate prognosis. When the number of calibration parameters is large, and the volume of computer simulation and observation data are also large, it brings significant challenges to both surrogate modeling and the associated Bayesian calibration. These challenges are addressed through three types of parallelization using the MapReduce technique. The first type of parallelization is pursued to efficiently collect simulation data at the training points for surrogate modeling. Next, the surrogate model training is parallelized using MapReduce. In the third step, parallelization of Markov Chain Monte Carlo (MCMC) technique is studied to efficiently perform Bayesian calibration in the presence of high-volume observation data. The proposed framework is implemented on the Spark platform. In addition to the parallelization of surrogate model training and Bayesian calibration, the singular value decomposition method is also employed to reduce the computational effort due to the high-volume data. The calibration of the thermal conductivity of concrete with field temperature observed from infrared thermography (IR) is used to demonstrate the proposed method.

The fourth objective investigates big data analytics in distribution surrogate modeling. In this objective, the training of a Gaussian mixture model (GMM) is parallelized via MapReduce. This provides the ability to efficiently build a high-dimensional surrogate model in the context of big data, which gives an analytical solution. This methodology will be illustrated by a mathematical example, as well as a thermal conductivity calibration example for a heterogeneous material.

1.3 Organization of the Dissertation

The subsequent chapters of this dissertation will be devoted to the objectives mentioned above.

Chapter 2 provides an introduction to the tools and methods needed for big data analytics in structural health monitoring. Structural health monitoring methods are reviewed first, and followed by big data techniques used for paralleling the computation. With respect to structural health monitoring, methods for data processing, diagnosis, and prognosis are introduced. Two surrogate modeling techniques (Gaussian process surrogate model and Gaussian mixture model) are reviewed. Among the big data techniques, MapReduce and Spark are explained.

Chapter 3 discusses the parallelization of data processing in structural health monitoring. Data processing is mainly used for diagnosis; here we focus on thermal image processing to draw inference about structural damage. However, the parallelization of thermal image processing can be easily generalized to other types of SHM data.

Chapter 4 extends the methodology in Chapter 3 to other steps in structural health monitoring, namely diagnosis (inverse problem) and prognosis (forward problem). Compared to Chapter 4, the diagnosis of structural damage status in Chapter 3 is deterministic, while the methodology developed in Chapter 4 includes uncertainty quantification.

Chapter 5 focuses on handling the model updating step for structural health prognosis, in the context of high-dimensional parameter space and large volume of data. By applying the methodology in this chapter, heterogeneous model parameters can be calibrated. This can reduce the spatial uncertainty in the model parameters, compared to considering homogeneous model parameters.

Chapter 6 addresses the distribution surrogate parallelization via MapReduce, which can help to build a full-size surrogate model, with high-dimensional inputs and outputs. Compared to the response surrogate used in Chapters 4 and 5, a distribution surrogate model can give an analytical solution, which makes model calibration or updating very fast. The parallelized distribution surrogate is implemented for the calibration of heterogeneous material properties.

Chapter 7 concludes the dissertation with a summary of accomplishments and directions for future research.

CHAPTER 2

BACKGROUND CONCEPTS AND METHODS

This chapter presents basic concepts and methods in structural health monitoring and big data analytics related to this study. First, we review the main steps of structural health monitoring, and focus particularly on image processing. Next, uncertainty quantification in structural diagnosis is reviewed, including the Bayesian approach and associated sampling methods such Markov chain Monte Carlo (MCMC) and Particle Filter (PF). The propagation of various uncertainty sources through the damage prognosis model to quantify the uncertainty in prognosis is reviewed next. In structural diagnosis and prognosis, repeated evaluation of physics-based numerical model (e.g., finite element model) is often required, which is expensive. Therefore surrogate modeling techniques are reviewed, which are applied in this dissertation. Since the goal of this study is to alleviate the computational burden in the above steps through big data techniques, the concept of MapReduce and its implementation in Spark are introduced. All the parallelization methods proposed in the subsequent chapters are realized in Spark using MapReduce.

2.1 Structural Health Monitoring

The purpose of structural health monitoring is to detect and diagnose damage in the structure, such that we can analyze future risk, predict the remaining useful life, and guide maintenance/repair actions if needed. In the context of damage diagnosis (Farrar et al., 2001), a four-step procedure is described: (1) Operational evaluation, (2) Data acquisition and cleansing, (3) Feature selection, and (4) Statistical model development. Operational evaluation defines what is to be monitored and how the monitoring process is to be implemented. Data acquisition and cleansing defines what data will be sampled and processed, and how the data will be sampled (i.e., in what frequency, how long it will be recorded, and how it will be preprocessed). The feature selection step defines the features that will be selected and the statistical distributions of the features. In the statistical model development step, the model is developed to detect the damage, predict remaining useful life, and quantify the uncertainty.

2.2 Image Processing

Digital image is one type of data format acquired in several SHM techniques, such as digital image correlation (DIC) and infra-red thermography. Damage is detected, located and quantified by comparing the image of the damaged structure against that for the intact structure, using image processing techniques. The general procedure described in (Baxes, 1994) is shown in Figure 2.1.

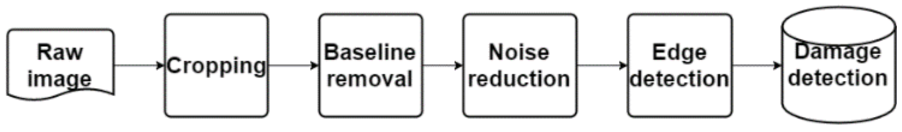


Figure 2.1 General procedure for image processing

After obtaining the raw image, preprocessing techniques (e.g. cropping, baseline removal and noise reduction) can be applied to prepare for edge detection, which can lead to damage detection. Noise reduction and edge detection are computationally expensive, and can benefit from the application of big data techniques.

2.3 Uncertainty Quantification of Structural Diagnosis

Various sources of uncertainty such as physical variability, data uncertainty, and model uncertainty affect structural diagnosis. The model inputs and parameters are physically variable in nature. System responses are measured through sensors, and the data may be noisy. Further, the sensors themselves may be damaged and wrongly imply deviation of system response from nominal behavior; the health monitoring system must distinguish such a scenario from the deviation caused due to actual damage in the system. These are the different aspects of data uncertainty. The models used for diagnosis are not accurate and are affected by model form assumptions and solution approximations. These different sources of uncertainty lead to uncertainty in the detection, localization, and quantification of damage. Therefore, the quantification of uncertainty in damage diagnosis is an essential step to guide decision making with respect to operations, maintenance, and risk management.

Classical statistics-based approaches for uncertainty quantification in damage diagnosis are limited with respect to data fusion, therefore this chapter uses a Bayesian approach for this purpose, which provides an efficient framework for updating the statistics as more data becomes available. Sankararaman and Mahadevan (2013) developed a Bayesian approach for uncertainty quantification in each of the three steps in damage or fault diagnosis, namely, detection, localization and quantification. Consider the estimation of uncertainty in damage quantification as an example. Bayesian updating is a statistical inference technique in which

Bayes' theorem is used to update the probability of a hypothesis as more information becomes available. Using Bayes' rule, the parameter updating process in structural diagnosis can be expressed as:

$$f"(q|y) = \frac{L(y,q)f\prime(q|y)}{\int L(y,q)f\prime(q|y)dq} \tag{2.1}$$

In Eq. (2.1), q is the true damage value, y is the detected damage. $L(y,q)$ is the likelihood function of q, and is proportional to $P(y|\, q)$, where $P(\cdot)$ means the probability density function. $f'(q|y)$ is the prior density function and represents the knowledge about q, while $f"(q|y)$ denotes the posterior probabilities when observations are available. Note that this is also the computation involved in Bayesian model calibration (i.e., estimation of model parameters based on available input-output data), which is often an important step in uncertainty quantification activities.

Often the construction of the posterior probability density function (PDF) is not analytically possible, thus sampling-based methods such as Monte Carlo Macro Chain (MCMC) and particle filter (PF) are commonly used to overcome this challenge. This chapter considers both techniques and discusses the methodology for fast computation later. The two techniques are briefly summarized below.

2.3.1 Markov Chain Monte Carlo Sampling

In Bayesian inference, where the objective is to compute the posterior distribution, MCMC sampling can be used to draw samples from the posterior distribution of a parameter of interest, and these samples can be used in conjunction with the kernel density estimation procedure to construct the posterior distribution. There are several popular MCMC algorithms, such as the Metropolis algorithm (Metropolis et al. 1953), Gibbs sampling (Roberts and Rosenthal 2006), and slice sampling (Neal 2003). We choose Metropolis algorithm in this dissertation as an example.

Assume that a function that is proportional to the PDF is readily available, as $f(x)$. For the purpose of illustration, consider the one-dimensional case, i.e. $x \in R$. The following steps constitute the algorithm in order to generate samples from the underlying PDF. Note that, the function $f(x)$ is always evaluated at two points and only the ratio is considered; the unknown proportionality constant is therefore cancelled.

Step 1. Set $i = 0$ and select a starting value x_0 such that $f(x_0) \neq 0$.

Step 2. Initialize the list of samples $X = x_0$.

Step 3. Repeat the following steps; each repetition yields a sample from the underlying PDF.

(a) Select a prospective candidate from the proposal density $q(x^*|x_i)$. The probability of accepting this sample is equal to $\frac{f(x^*)}{f(x_i)}$.

(b) Calculate acceptance ratio $\alpha = \min(q, \frac{f(x^*)}{f(x_i)})$.

(c) Select a random number u, uniformly distributed on $[0, 1]$.

(d) If $u < \alpha$, then set $x_{i+1} = x^*$, otherwise set $x_{i+1} = x_i$.

(e) Augment the list of samples in X by x_{i+1}.

(f) Increment i, i.e. $i = i + 1$.

The Metropolis algorithm assumes that the proposal density is symmetric (to ensure the state transition is reversible), i.e.$q(x^*|x_i) = q(x_i|x^*)$, and a usual choice is to let $q(x^*|x_i)$ be a Gaussian distribution centered at x_i. After the Markov chain converges, the samples in X can be used to construct the posterior PDF of X using kernel density estimation. The common practice is to generate hundreds of thousands of samples and discard the first few thousand samples to ensure that the samples considered for the posterior distribution are only those after the Markov chain has converged.

2.3.2 Particle Filter

Particle Filter, also known as Sequential Monte Carlo (SMC), is a method used for approximating the posterior distribution of the quantity of interest. The key idea is to represent the required posterior density function by a set of random samples (particles) with associated weights, and to compute the estimates based on these samples and weights. Let $\boldsymbol{X}_{0:k_i}, i = 0, \cdots, N$ be particles with associated weights $\boldsymbol{W}_k^i, i = 0, \cdots, N$, where N is the number of particles, and k is the state index. The posterior density at time t_k can be expressed as:

$$\pi(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N} w_k^i \delta(x_{0:k} - x_{0:k}^i) \tag{2.2}$$

The main steps are summarized below (Orlande et al. 2011):

Step 1. For $i = 1, \cdots, N$ draw new particles x_k^i from the prior density $\pi(x_k|x_{k-1}^i)$ and then use the likelihood density to calculate the correspondent weights $w_i^k = \pi(z_k|x_k^i)$.

Step 2. Calculate the total weight $T_w = \sum_{i=1}^{N} w_k^i$ and then normalize the particle weights.

Step 3. Resample the particles as follows:

Step 3.1. Construct the cumulative sum of weights (CSW) by computing $c_i = c_{i-1} + w_k^i$ for $i = 1, \cdots, N$, with $c_0 = 0$.

Step 3.2. Let $i = 1$ and draw a starting point u_1 from the uniform distribution $U[0, N^{-1}]$.

Step 3.3. For $j = 1, \cdots, N$

(a) Move along the CSW by making $u_j = u_i + N^{-1}(j - 1)$.

(b) While $u_j > c_i$, make $i = i + 1$.

(c) Assign samples $x_k^j = x_k^i$.

(d) Assign weights $w_k^j = N^{-1}$.

Compared to MCMC, PF does not have the two disadvantages as: 1, correlated samples which could be solved via thinning (pick one sample for every k samples); and 2, necessary burn-in period (dropping first m samples) at the beginning. Both of those two problems lead to a waste of samples in MCMC. Furthermore, PF has several other advantages such as: 1, scaled well to high dimensional problem; 2, more efficient compared to MCMC; and 3, easier to implement. On the other hand, there are drawbacks, and the most important one is the problem of lacking of diversity, in other words, once a state loses particles, it cannot regain them without motion. Techniques such as Rao-Blackwellization (Doucet et al. 2000) can help to fix this issue.

2.4 Uncertainty Quantification for Structural Prognosis

Similar to diagnosis, structural prognosis (forward problem) is also affected by both aleatory and epistemic uncertainty sources. Due to insufficient information, epistemic uncertainty may arise about the exact values of deterministic model inputs or the distribution characteristics of stochastic model inputs. Another type of epistemic uncertainty is model uncertainty. Model uncertainty represents the inability of the model to accurately represent the true physical behavior of the system. Uncertainty due to a model may be due to three sources: (1) lack of knowledge about the precise values of model parameters, due to limited data; (2) numerical solution errors that arise from the methodology adopted in solving the model equations; and (3) model form errors, which arise due to assumptions and simplifications made in the development of the models. Calibration, verification and validation are the activities that can be used to quantify the three sources of uncertainty. A Bayesian approach for the aggregation of various uncertainty sources as well as the aggregation of results of model calibration, verification and validation towards uncertainty quantification in the system response prediction was developed by Sankararaman and Mahadevan (2015), and was further extended to reliability analysis by Nannapaneni and Mahadevan (2016).

Consider a generic prognosis model $Y = G(\boldsymbol{X})$, which is used to represent the degradation of an engineering system. The input is a vector and hence denoted in bold as $\boldsymbol{X}$, whereas the output Y is a scalar. The model G is deterministic, i.e. for a given realization of $\boldsymbol{X}$, there is a corresponding output, which is a realization of Y. The inputs $\boldsymbol{X}$ are uncertain, and this leads to uncertainty in the output Y. A generic realization of $\boldsymbol{X}$ is denoted as $\boldsymbol{x}$, and a generic realization of Y is denoted as y. The goal in uncertainty propagation is to propagate the input uncertainty through G, in order to the calculate the CDF $F_Y(y)$. The CDF of Y can be calculated as:

$$F_Y(y) = \int_{G(x)<y} f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x} \tag{2.3}$$

where $f_{\boldsymbol{X}}(\boldsymbol{x})$ is the probability distribution of $\boldsymbol{X}$. The PDF can be calculated by differentiating the CDF, as:

$$f_Y(y) = \frac{dF_Y(y)}{dy} \tag{2.4}$$

Note that prognosis and reliability analysis have similar types of computation, namely uncertainty propagation. The distinction between the two is that prognosis is for a particular structure, thus its properties are unique; whereas in the case of reliability analysis we also need to consider variability across multiple realizations of the structural properties (model parameters). In both types of computation, if there is statistical uncertainty regarding the distribution parameters of the input random variables, this creates a family of distributions for the input and therefore the output. On the other hand, model errors can be included in the uncertainty propagation as additive error terms, quantified using calibration, verification and validation activities and represented using probability distributions. The aggregation of various types of uncertainty in the uncertainty propagation analysis is effectively done through Monte Carlo simulation. However, Monte Carlo simulation is expensive, thus the next section explores the use of MapReduce to parallelize the uncertainty propagation in the forward problem. Nannapaneni and Mahadevan (2016) also explored a FORM-based strategy for faster computation, but found it to be of limited use in the presence of nonlinearities and uncertainty regarding correlations.

2.5 Surrogate Modeling

As mentioned in Sec. 1.1, there are two types of surrogate models: response surrogate and distribution surrogate. Response surrogate modeling techniques have been extensively investigated in the literature, such as polynomial chaos expansion (Ghanem & Spanos, 1990), polynomial response surface (Rajashekhar & Ellingwood, 1993), support vector regression (Boser et al., 1992), relevance vector regression (Tipping, 2001), and Gaussian process (GP) interpolation (Rasmussen, 2006; Santner et al. 2013; Bichon et al., 2008). On the other hand, Bayesian network (Jensen, 1996; Heckerman, 1998) is a general form of distribution surrogate, while there are some approximations such as multivariate Gaussian (Rose and Smith, 1996), Gaussian copula (Nelsen, 1999; Liang and Mahadevan, 2016) and Gaussian mixture model (Reynolds, 2015). All three approximate distribution surrogate models give fast, analytical solutions; among these, the Gaussian mixture model is the most accurate but also takes much longer time to train. In this dissertation, one response surrogate model (Gaussian process) and one distribution surrogate model (Gaussian mixture) are used, which are discussed in detail below.

2.5.1 Gaussian Process Surrogate Model

Since Bayesian updating requires repeated runs of computer model, an inexpensive surrogate model is often used in this analysis instead of the original model to reduce the computational cost. Many types of surrogate modeling techniques are available; Gaussian process surrogate model is chosen in this section for this the purpose of illustration (Rasmussen 2006).

A Gaussian process is specified by its mean function and covariance function and is a generalization of the multivariate normal distribution. We define the mean function m(x) and the covariance function $K(x,x')$ of a random process $f(x)$ as $m(x) = E[f(x)]$ and $K[(f(x) - m(x))(f(x') - m(x'))]$ respectively. The process $f(x)$ can then be denoted as $f(x) \sim GP(m(x), K(x,x'))$. In prediction, the joint distribution of the training outputs yT and the prediction yP is:

$$\begin{bmatrix} y_T \\ y_P \end{bmatrix} \sim \left(\begin{bmatrix} m_T \\ m_P \end{bmatrix}, \begin{bmatrix} k_{TT} & k_{TP} \\ k_{PT} & k_{PP} \end{bmatrix} \right) \tag{2.5}$$

where T indicates training and P indicates prediction. The prediction conditioned on the training points follows a Gaussian distribution $y_P \,|y_T \sim N(m,S)$, in which, $m = K_{PT}K_{TT}^{-1}y_T$, and $S = K_{PP} - K_{TT}^{-1}K_{PT}^{T}$.

A number of common functions can be used as kernels to construct the covariance matrices. As an example, the commonly used squared exponential function is used here:

$$K(x_i, x_j) = \sigma^2 e^{-\frac{1}{2}\left(\frac{x_i - x_j}{l}\right)^2} \tag{2.6}$$

in which l is the length scale (which controls the correlation decay with distance) and σ^2 is the magnitude of variance. Based on the training data, these parameters can be estimated by the maximum-likelihood estimation (MLE) method.

2.5.2 Gaussian Mixture Model

The Gaussian mixture model (Bishop, 2006) is a simple linear combination of Gaussian components, which can provide a richer class of density models than a single Gaussian. The

Gaussian mixture distribution can be written as

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.7}$$

The weights and parameters of the component distributions can be obtained by maximizing the likelihood. However, likelihood maximization requires the derivatives of the likelihood function with respect to all the unknown values, the parameters and the latent variables, and simultaneously solving the resulting equations. In statistical models with a large number of unknown variables such as GMM, this is usually impossible. Expectation-Maximization (EM) is a powerful algorithm for finding maximum likelihood solutions (Dempster et al., 1977; McLachlan & Krishnan, 1997). The main steps of EM for GMM are listed below (Bishop, 2006):

Step 1. Initialize the means $\boldsymbol{\mu}_k$, covariance $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.

Step 2. **E-step**. Evaluate the posterior distributions using the current parameter values

$$\gamma(z_{n,k}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.8}$$

Step 3. **M-step**. Re-estimate the parameters using the current posterior

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{n,k}) \boldsymbol{x}_n \tag{2.9}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{n,k}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})^T \tag{2.10}$$

$$\boldsymbol{\pi}_k^{new} = \frac{N_k}{N} \tag{2.11}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{n,k}) \tag{2.12}$$

Step 4. Evaluate the log likelihood

$$\ln p(X|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} ln\{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k)\} \tag{2.13}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied, return to step 2. An example threshold for the difference of the log likelihood between this step and last step could be 1×10^{-3}.

2.6 Big Data Analytics

There are two different directions to pursue in solving the big data problem. First, when the data is too large to process, in order to reduce the computational cost, it may sometimes be desirable to compress the data before processing. Data compressed into feature vectors can help to reduce the dimension of data, by exploiting statistical redundancy of the raw data (Sohn et al., 2001). Additionally, another kind of reduction can be achieved via reducing the data size using samples of the data, known as compressive sensing. One example is the compressive sampling of accelerometer signals (Bao et al., 2010). While it seems to be a reasonable way to handle the voluminous data, one of the issues in data compression is the reduced accuracy of the detection, which sometimes leads to the low quality of the structural health monitoring, resulting in unreliable decision making.

In contrast to data compression, the second option, namely parallel and distributed computing offer alternatives to reduce the time cost of data analytics, without causing any precision loss. Parallel computing is more tightly connected to multi-threading, or how to make full use of a single CPU; Distributed computing refers to the notion of divide and conquer, executing subtasks on different machines and then merging the results.

Theoretically, distributed computing is much more powerful, since more memory and CPU resources (from the cluster) are available, although the bandwidth among the connected computers can sometimes become the main limitation. Message Passing Interface (MPI) is one of the most popular distributed computing methods used for a long time, and applications can be found in structural health monitoring (Kiepert & Loo, 2012, Chakraborty et al., 2009). MPI's goals are high performance, scalability, and portability. Another conceptually similar approach in the context of big data is MapReduce. Utilizing a cluster of nodes, MapReduce performs two essential functions – it assigns work to various nodes within the cluster, and then organizes and reduces the results from each node into a cohesive answer to a query (Dean & Ghemawat, 2008).

Although the main purpose of both MPI and MapReduce is to improve the efficiency via parallelization, there are several differences between them. First, MPI is designed to handle large amounts of data exchange between computers, while MapReduce focuses on embarrassingly parallel implementation (no much information exchange among computers). Second, MPI is appropriate for iterative algorithms that are computationally expensive, whereas MapReduce is fit for the case where the expense is mainly caused by the data itself. Third, although MPI can also be built to be scalable and fault tolerant, it needs much effort to ensure the performance and reliability of such a system, MapReduce on the other hand, is created to be easily scalable and fault-tolerant. A detailed discussion about the relationship between MPI and MapReduce can be found in (Chen et al., 2011).

2.6.1 MapReduce framework

MapReduce is a framework designed for processing large datasets, by utilizing multiple nodes (machines) for the computations. It takes key/value pairs as inputs and generates other key/value pairs as outputs. As mentioned earlier, the MapReduce framework can be split into two steps: map and reduce, both of which are created by the user. Before applying the MapReduce model, the user will need to write the input as the key/value pair. The key/value pair (k1, v1) will then be input to the map function, which will generate the intermediate key/value pairs (k2, v2). Then the intermediate key/value pairs are passed to the reduce function, which merges together these values to form a smaller set of values. This process allows to handle lists with high memory requirements and is displayed in Fig. 2.2.

```
map(k1, v1) − > list(k2, v2)
reduce(k2, list(v2)) − > list(v3)
```

Figure 2.2 MapReduce process to handle lists

A cluster of computers (nodes) are used to implement this framework (Figure 2.3). One of them is the master node and the others are slave nodes. As shown in Figure 2.3, the master node talks to the user program, and assigns the tasks to the slave nodes (workers). First, the input files are parsed and split into smaller pieces (size 16MB to 64MB). The master will select the idle workers and assign each a map task or reduce task. Then each worker will do its own task and when all tasks are completed, the output files will be collected and synthesized by the master node.

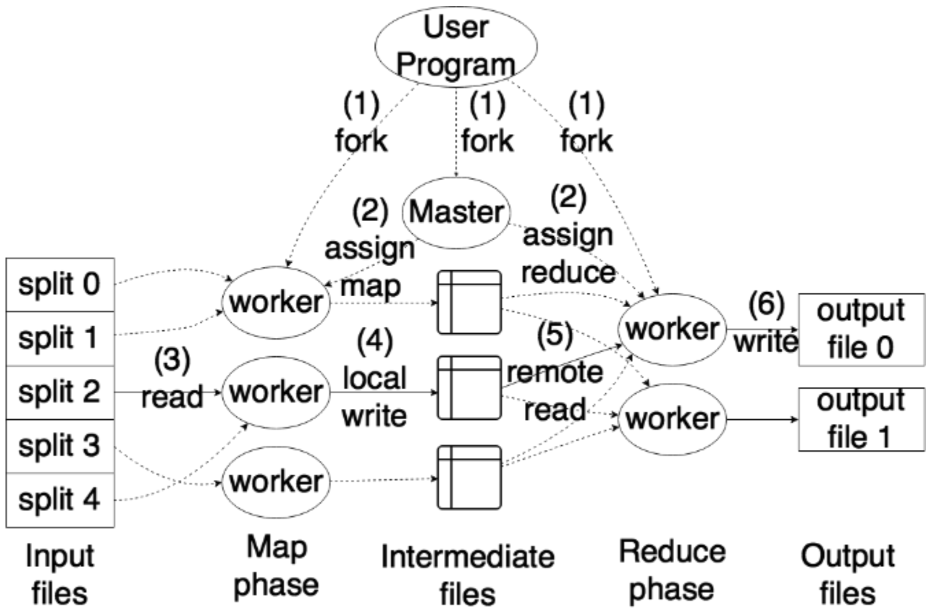


Figure 2.3 MapReduce execution overview

2.6.2 Spark

While there are different implementations of MapReduce, Apache Spark (Zaharia et al. 2012) is the one chosen in this study. Spark is an open source cluster computing framework. APIs (Application Program Interface) for Java, Scala and Python are available, which is convenient for non-computer science programmers. Beside the basic capability of using the MapReduce methodology, Spark employs Resilient Distributed Datasets (RDD) that enable efficient data reuse in a broad range of applications. Furthermore, in contrast to other systems, Spark applies coarse-grained transformations (e.g., map, filter and join) to allow for the fault-tolerance feature. In contrast with fine-grained transformation, the coarse-grained transformation is applied on the entire dataset, instead of on a single data point. Instead of storing the actual data, the logging of the transformation can ensure that there is enough information to redo the operation if an RDD is lost. Due to the adoption of RDD, iterations in the computational algorithm do not need to repeatedly execute the reading and writing operations on the file system; this greatly reduces the computational cost in iterative algorithms (Fig. 2.4).

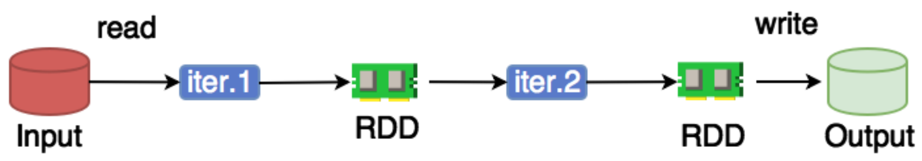


Figure 2.4 RDD in Spark

CHAPTER 3

BIG DATA ANALYTICS IN DATA PROCESSING

3.1 Structure, Sensors and Data Acquisition

When numerous images (Gigabytes or Terabytes of data) are collected in structural health monitoring, the data is too large and a traditional data processing framework (storage, processing and manipulating) is not feasible; therefore a big data analytics framework needs to be employed. The methodology to apply the big data technique in health monitoring will be developed in detail in this section. Structural health monitoring systems have the following elements: structure, sensors, data acquisition system, data transfer and storage mechanisms, data processing, and data manipulation. Each element's relation to big data are discussed below. A large volume of data can be caused by the size of the structure being monitored, or by the number of sensors. The structure gives the scope, and the sensors give the resolution.

In SHM, the engineering structure is the target to be monitored and regarding which the decision needs to be made (whether to use, maintain, repair or retire the structure based on the diagnosis result). For example, suppose instead of the piers of the bridge to be monitored, the health of the whole bridge (deck, load-carrying elements, piers, and foundations) is being evaluated, with the processing ability of big data. In this case, the resolution is not changing, but the data volume is greatly enlarged.

As mentioned earlier, another cause of big data in SHM is resolution. Similar to the monitoring scope, the number of sensors can be increased with the data processing ability provided by the big data techniques. With more sensors used in monitoring, more information will be available for analysis.

In the monitoring process, data will be generated by sensors, and then interpreted and transferred to the data processing computer, via data acquisition system (DAQ). The sampling rate is controlled by the DAQ device, which directly affects the resolution and data size. After acquired by the DAQ device, the data is stored in the computer (either a laptop or a desktop) connected with the DAQ device. The next step is to transfer the data to the cluster. For the Linux or Mac operating system, the command for data uploading is ‘scp’. The syntax of ‘scp’ is given in Fig. 3.1.

```
scp -r /local/path/to/foo
user@your.server.example.com: /cluster/path/to/foo
```

Figure 3.1 Scp syntax for data uploading

In Fig. 3.1, the syntax /local/user/path/to/foo indicates the local folder, while user@your.server.example.com:/cluster/path/to/foo indicates the target folder in the cluster, and -r implies recursive copying of the files in the folder. ‘foo’ is commonly used as a placeholder name. When the operating system for the client computer is Windows, a similar command can be used after installing WinSCP or PuTTY. The transferring speed is limited by the devices on both ends, and by the bandwidth of the connection between the client and cluster.

Normally the MapReduce application is automatically paired with the corresponding file system, such as Hadoop with HDFS (Hadoop Distributed File System), Amazon EMR with Amazon S3, and Windows Azure and WASB (Windows Azure Storage Blobs). However, the user can also choose a different file system other than the default paired one, when it is more applicable to do so. For example, here we use Spark, paired with GPFS (General Parallel File System). Additionally, the distributed file system will divide the large data file into blocks (normally $64\ MB$ to $128\ MB$, and normally the user is allowed to change the block size in the actual application of MapReduce).

3.2 Data Processing

As reviewed previously, there might be different data formats to be processed in structural health monitoring. Here we consider thermal image processing as an example. The common procedure for processing digital images is: cropping, baseline removal, noise cancellation and feature extraction. Each image is composed by pixels (Fig. 3.10 for example), where each pixel represents the temperature of the location.

3.2.1 Baseline Removal

Baseline removal subtracts pixel values by the corresponding pixel from an image of the control group. It happens when the control group is available. This can enhance signal characteristics for diagnosis.

3.2.2 Cropping

The cropping is realized by only storing and plotting the corresponding part of the target structure we analyze. Compared with the raw image, the temperature contour of cropped image is zoomed in (Fig. 3.11). Normally since in the observation procedure, of the locations of the structure and camera do not change, the cropping pixel range for all the images is the same.

3.2.3 Noise Cancellation

Uniform filtering is used for the purpose of noise cancellation. The basic idea is to average each pixel by the value of adjacent pixels. Notice that uniform filtering is different from simple moving average (SMA), in that uniform filter is doing averaging by putting the target point in the center while SMA is doing biased averaging. Mathematically, the uniform filtering process is basically a 2D convolution operation. To illustrate the convolution operation, the 1D convolution operator formula is defined in Eq. (3.1), in which f is the uniform kernel, and g is the image matrix to be operated on. The kernel can be of different sizes, and Fig. 3.2 shows how a kernel with size 3×3 works on a 5×5 target matrix. To perform convolution, first align the center element of the kernel matrix with the element on the target matrix, and then sum up the multiplication between all aligned element-pair. For example, the convolution on the element $(1, 1)$ is 7.67, as is shown in Fig. 3.2. Move the kernel along x and y axis until convolution of all elements are carried out. Refer (Jain et al., 1995) for detailed implementation. After the uniform filtering, the image is smoothed, i.e., more continuous everywhere (Fig. 3.12).

$$
\begin{aligned}
m(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \\
&= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau
\end{aligned} \tag{3.1}
$$

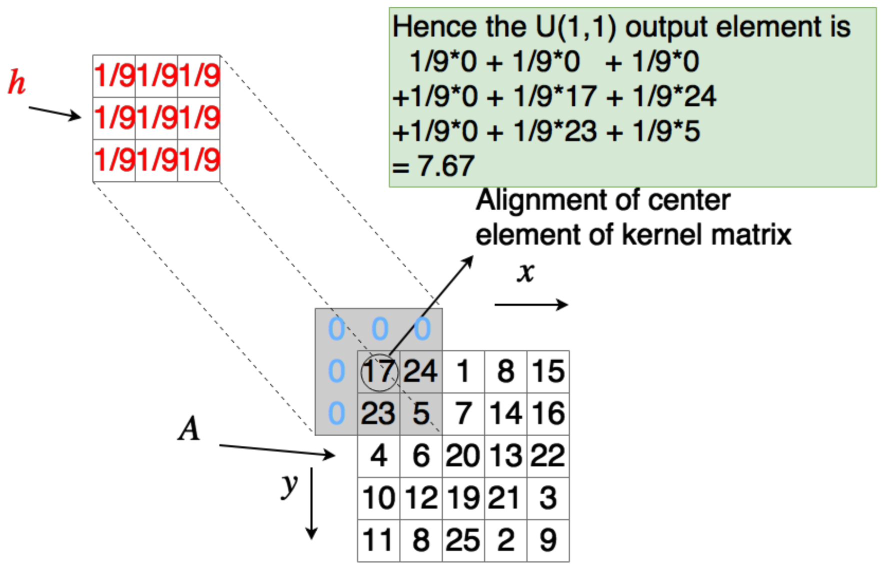


Figure 3.2 Uniform filtering example

3.2.4 Feature Extraction

The Sobel filter method (Jain et al., 1995) is used here for the feature extraction, based on the image obtained after uniform filtering. The other edge detection algorithms such as Canny, Prewitt, Robert, Laplacian and Laplacian of Gaussian filters were tried and found that Sobel filtering performed best in our problem. The selection of algorithm would be problem-dependent and any desired algorithm can be plugged in the big data analytics framework in the same way as Sobel filter.

The basic idea behind Sobel filter is similar to the uniform filter, which is also a 2D convolution operation, where the only difference is the filter kernel. Similar to the uniform filter, Sobel filter can also be performed with different sizes. The difference is that for uniform filtering, there is only one kernel, which is a $n \times n$ matrix filled with the value $1/n^2$. For Sobel filtering, the filters for x and y directions can be different (Fig. 3.3). Additionally, the kernel can be split into the product of two 1D kernels, for averaging and differencing in two directions (Fig. 3.4). To differentiate the damaged area, gradient ranges in both x and y directions are needed, and thresholds is applied to detect the edges of damages.

$$h_x = \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \qquad\qquad h_y = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

(a) (b)

Figure 3.3 Sobel filter kernels: (a) kernel for x direction, and (b) kernel for y direction

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline \end{array}$$

Figure 3.4 Split of Sobel filter kernel (x direction) into averaging and differencing

3.2.5 MapReduce for Data Processing

The basic idea of the application of data processing in MapReduce is to divide the files into different partitions (each partition contains multiple files), and then perform the mapping and reducing operations separately. To fully use the resources, the number of partitions is always greater than the number of instances (i.e. cores, of which each node might contain a multiple). For example, if the number of files to be analyzed is 100, and the number of cores available is 20, the number of partitions should be at least 20. Otherwise some of the cores will be idle.

In structural health monitoring, the data is normally sampled as separate files (images or signals). For each image and signal, a separate processed result is obtained, without combination (Fig. 3.6). In that case, the Reduce function is omitted, and only the Map function remains. All the data processing functions on the assigned files are combined within a single Map function. The Map function is defined by the user, in which the reading, processing, and writing functions are all included, as shown by the pseudocode below:

```
Pseudocode 3.1:
Map(x):
    function InputData = ReadData(x)
    function OutputData = Processing(InputData)
    function WriteData(OutputData)
    return (x, 0)
SparkContext(appName=”myApp”).parallelize(Filelist,
N).map(mapper).count()
```

Figure 3.5 Pseudocode of MapReduce implementation for data processing

The pseudocode in Fig. 3.5 has two steps. First, a *Map* function is defined (mapper), within which all the actual data processing functions are defined (reading, filtering, writing). The argument x is the file to be analyzed, which is assigned by the task manager. As discussed previously, since there is only the *Map* function, the input file can be mapped with any value (here we mapped x to 0). The reason it can be any value is here we only use the *Map* function to trigger the parallelization, without caring for the output of the *Map* function. The second step, SparkContext, represents the connection to the cluster, which is the main class in Spark; parallelize is the method to split the input files into *N* partitions; and 'map' is the method to call the Map function defined in the first step and to pass the input file to it. The count method is used to count the number of outputs. The number of outputs is not of interest, since the result has already been obtained in the *Map* function. However, it is needed since the transformations (parallelize, map) only created the RDD instance, which needs some actions to execute it.

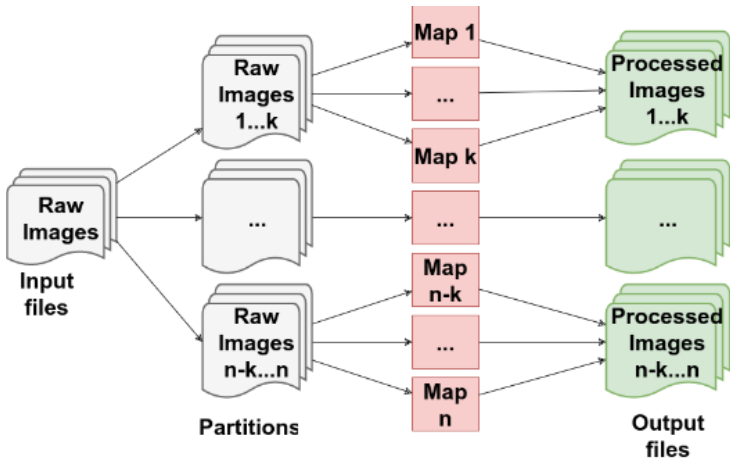


Figure 3.6 Schematic description of the MapReduce process for data processing

After the cluster finishes all the tasks, the results are stored in the designated directory defined in the ‘WriteData’ function. Then the next step is to retrieve the data files from the cluster to the local computer, since normally it is not convenient to visualize the data remotely on the cluster. To transfer data back from the cluster, the user can use the ‘scp’ command similar to the one used for transferring the data to the cluster.

Operations for image processing (cropping, uniform filtering, and Sobel filtering) need to be applied on all the images, with all parameters (cropping range, uniform filtering kernel size, and Sobel filtering gradient cutoff) remaining unchanged. As defined earlier, the reading, writing and processing functions are all included within the Map function. There are three sub-functions: ‘Cropping’, ‘UniformFilter’ and ‘SobelFilter’.

Several remarks about the processing function are in order. First, the input data is no longer a key/value pair but is an actual image (pixel matrix). Second, the sub functions inside will be sequentially executed, since the outputs of each sub function will be fed into the next sub function as inputs. Third, the sub functions (‘Cropping’, ‘UniformFilter’, and ‘SobelFilter’) can be replaced easily with other functions according to the actual data processing task.

In summary, the steps for the big data analytics of image processing in structural health monitoring are: (1) upload the acquired data from the local computer to clusters; (2) prepare the image processing functions, and substitute into the Map function shown in Fig. 3.5; and (3) run Spark to process and retrieve the data files from the cluster back to the local computer.

3.3 Numerical Example

This example illustrates the basic application of big data analytics in structural health monitoring. The purpose of the monitoring in this example is to detect holes drilled into a $15.5\ in \times 15.5\ in \times 2\ in$ concrete slab (Fig. 3.7) using infrared thermography imaging. Holes of $5/8\ in$, $1/2\ in$, and $5/16\ in$ diameter (all of them are $4.45\ in$ deep) were drilled into the side of the concrete slab, as shown in Fig. 3.8. The holes are required to be detected by the monitoring technique in this example.

Since the focus of this example is the application of big data technique to structural diagnosis, we use the holes only to illustrate this capability. In this case, the ground truth is known, which facilitates performance evaluation of the monitoring technique. In realistic situations, concrete damage could be of many types (physical, chemical, and mechanical), due to various causes such as freeze-thaw, chloride penetration, alkali-silica reaction etc. Temperature, humidity, and the properties of the concrete constituents (cement, aggregates, reinforcing steel, water content, and chemical admixtures) play a crucial role in the evolution of various types of damage. Damage in concrete eventually manifests as cracks, delamination, spalling etc., and the edge detection approach illustrated here could be applied to different situations.

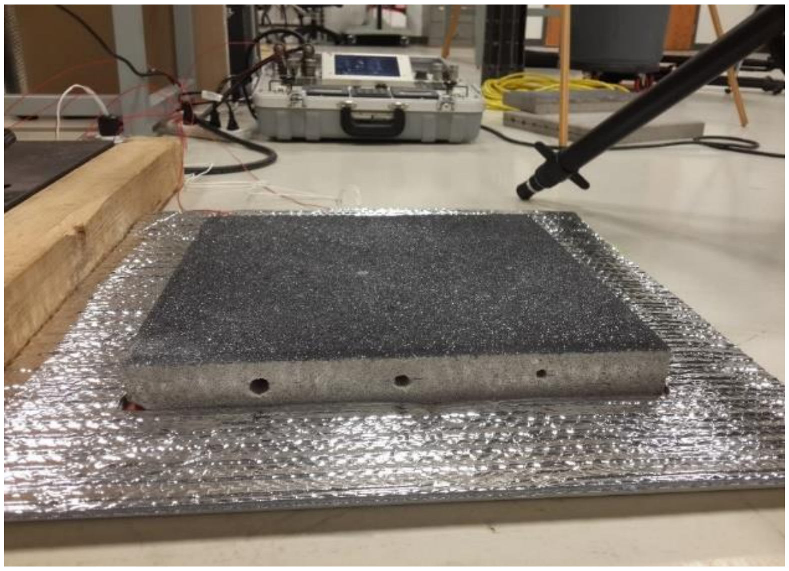
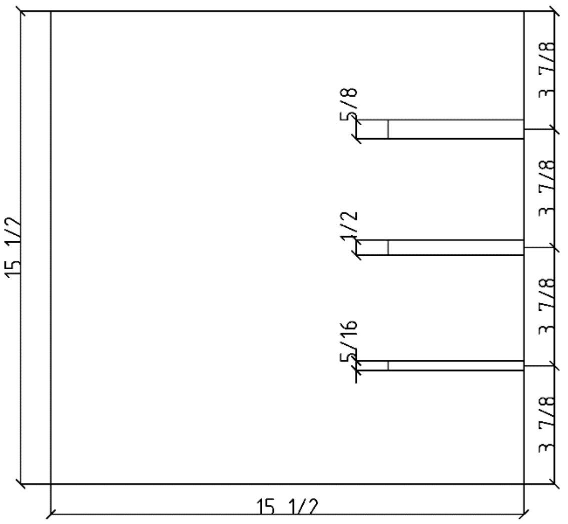
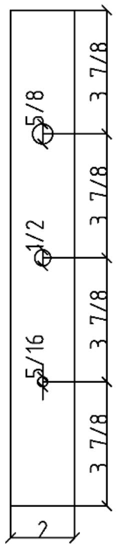


Figure 3.7 Thermography camera and the specimen to be monitored



(a)



(b)

Figure 3.8 Sketch of the specimen (a) top view (b) side view

3.3.1 Experiment Setup

The mechanics of damage detection using infrared thermography is based on the differences in heat transfer properties of different materials. The air in the drilled holes in the structure has much lower thermal conductivity coefficient than concrete, which will lead to a lagging phenomenon, i.e., the heating and cooling time of the hole are slower than the surrounding solid region. The slab is placed on a HEATCON thermal blanket and uniformly heated from below. The infrared thermography camera can detect the temperature of the surface of the slab (Fig. 3.7, Fig. 3.8) and store the temperature values as images via the DAQ system. We also place reflective material around the slab, in order to prevent direct heat transfer from the thermal blanket to the air around the slab; thus the thermal camera detects the temperature change on the top surface slab mainly caused by the heat transfer from the blanket through the slab.

3.3.2 Thermal Loading

Each thermal cycle has a total duration of 70 minutes. The heating profile is shown in Fig. 3.9. A HEATCON composite system controller was connected to the thermal blanket and used to program a defined thermal cycle that can be repeated as many times as needed for a test. Two thermocouples were used to measure and monitor the heat applied by thermal blanket. One thermocouple was placed beneath the blanket and the other thermocouple was placed between the thermal blanket and the concrete sample (Fig. 3.10).

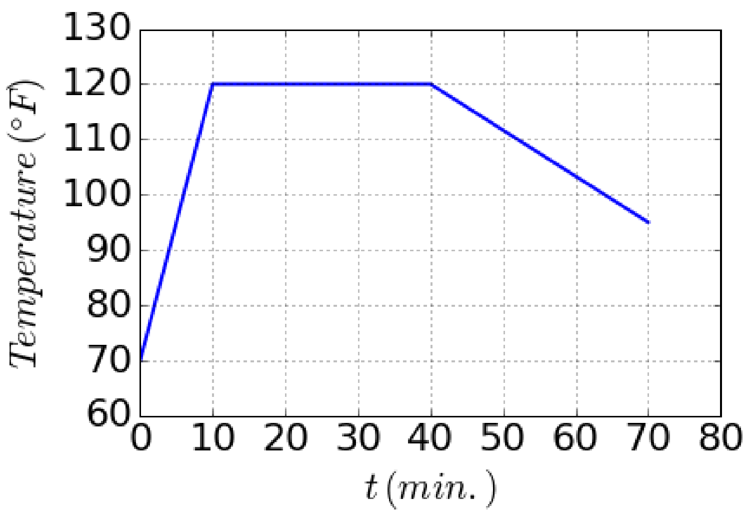


Figure 3.9 Thermal loading time history (scaled values)

For thermographic imaging, a FLIR Infrared (IR) camera is used to detect the temperature contours on the surface of the concrete slab. These contours can be analyzed to detect flaws or defects inside the slab that cannot be easily detected by visual inspection. The

FLIR IR camera was setup to capture images of the concrete slab every 1 second.

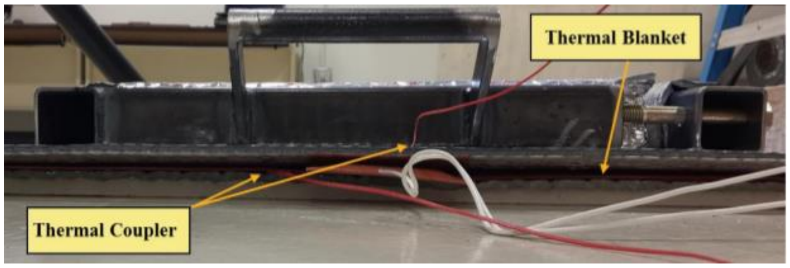


Figure 3.10 Thermal blanket and thermo couple

3.3.3 Data Acquisition System

The FLIR IR software is an integrated environment that allows the user to configure the sampling rate, resolution, and storage. Also the software can visualize the current captured image, and store the images in the designated path in the '.tls' format, which is specially used by this software.

3.3.4 Data Transfer and Storage Mechanism

After the sampling is completed, the data stored in the file *.tls can be exported in different format, such as .csv, .m, .txt, .jpeg. In this study, we used .csv to represent each image. For the heat loading period considered, 4231 images were sampled, and the total size is 19.4 GB. The '.tls' file is stored in the computer connected with the DAQ system, and the size is much smaller. The exported .csv files were stored in a portable drive, through which they were transferred to the analysis computer client. In order to use MapReduce to analyze the data, the data was uploaded to the cluster, which in this case was located within ACCRE

(Advanced Computing Center for Research and Education) at Vanderbilt University.

3.3.5 Data Processing

The implementation of various steps in processing the thermal image data are discussed in detail and the results are presented below.

3.3.5.1 Baseline Removal

As reviewed previously, the common procedure for processing digital images consists of: cropping, baseline removal, noise cancellation and feature extraction. In this example, results can be obtained without control group. Thus there is no baseline removal needed here. This can save almost half the cost of data storage. For each image, the resolution is 640×512 pixels (Fig. 3.11).

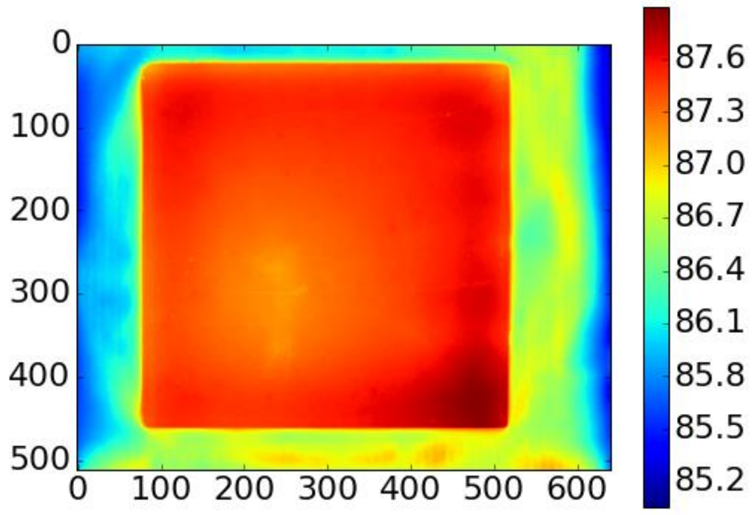


Figure 3.11 Example of raw image before cropping ($t = 2835$ s)

3.3.5.2 Cropping

Fig. 3.11 shows the raw thermography image of the top surface of the slab and reflective material, 2835 seconds after start of the heating. Notice that the area corresponding to the slab has much higher temperature compared to the surrounding reflective material. Thus the image needs to be cropped in order to achieve greater resolution in analyzing the temperature distribution within the slab. After several trials, the appropriate pixel range for cropping was found to be $[83:518, 25:460]$. The cropped image is shown in Fig. 3.12.

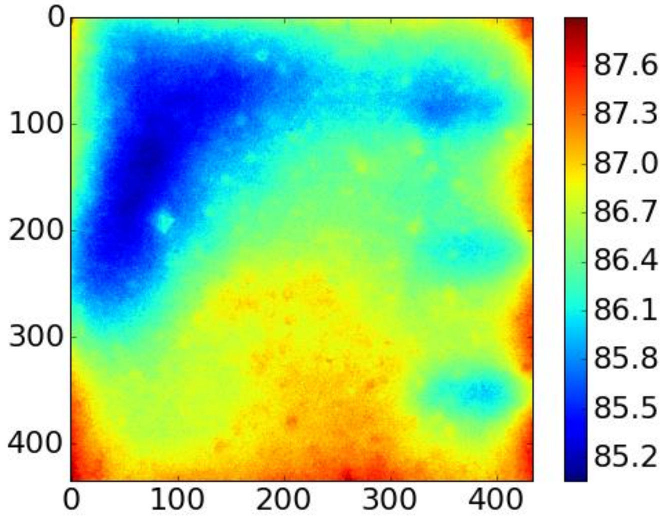


Figure 3.12 Cropped image ($t = 2835$ s)

The image shows boundary effects, where additional heat may be introduced from the area around the slab, since the reflective material may not block all of the heat from the thermal blanket, especially since there was a small gap between the slab and the reflective material. It is also seen that there is a large area on the upper left quadrant, where the temperature is low. It may be due to the non-uniformity of the heating setup (such as lack of contact between slab and blanket), and heterogeneity of the concrete slab; the feature extraction step will reveal whether these effects are significant. As explained in the methodology section, the cropping pixel range for all the images are the same.

3.3.5.3 Noise Cancellation

A 22×22 kernel uniform filtering is used for noise cancellation, as shown in Fig. 3.13. It can be observed that after the uniform filtering, the image is smoother. By doing this, the noise in the image is greatly reduced. Note that Fig. 3.13 roughly indicates the three holes in the right hand side. There is also a large, low temperature area on the left, but this gets eliminated in the subsequent feature extraction step.

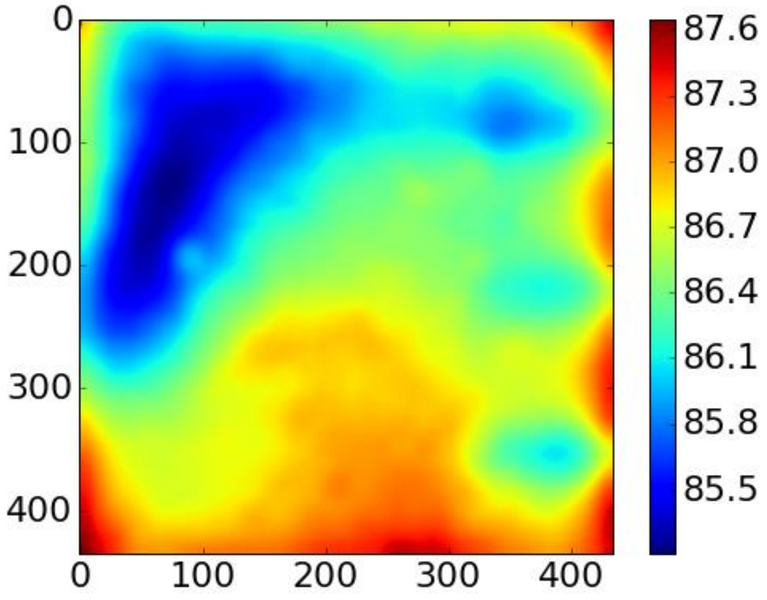
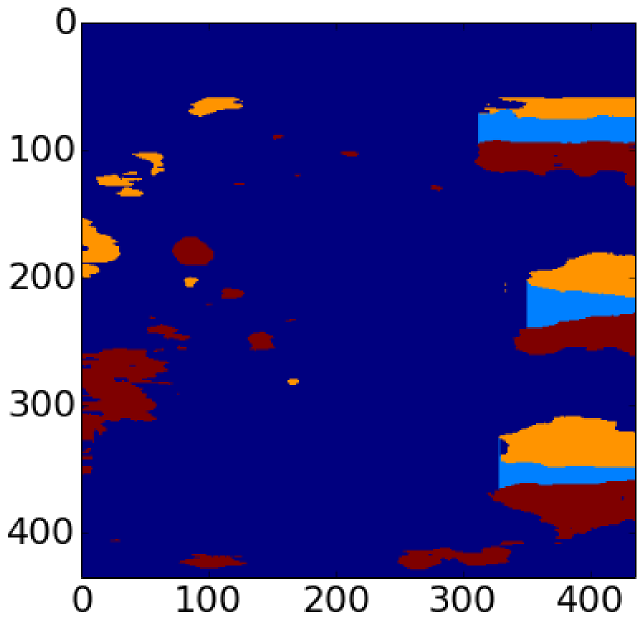


Figure 3.13 Image after uniform filtering ($t = 2835\ s$; 22×22 kernel)

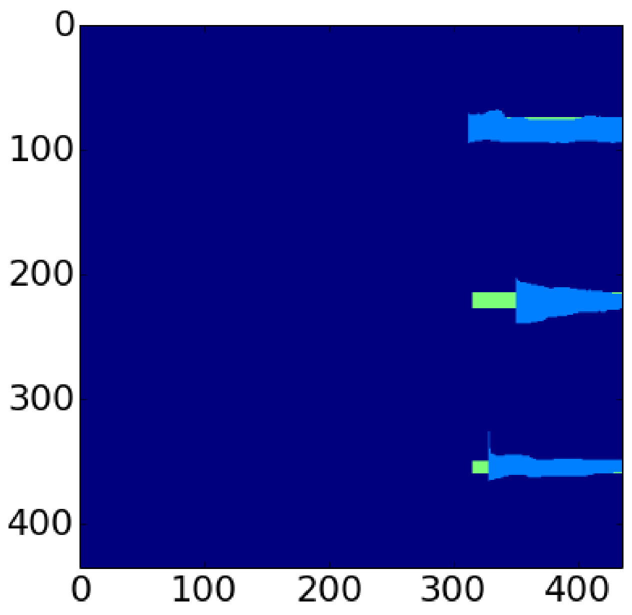
3.3.5.4 Feature Extraction

Sobel filter is used for the feature extraction, based on the image obtained after uniform filtering. After applying Sobel filtering, the image shows the detected holes in the slab (Fig.

3.14 (a)). The holes are detected by first obtaining the upper edges (yellow region on the right hand side in Fig. 3.14 (a)) and lower edges (red region on the right hand side in Fig. 3.14 (a)), and then plot the region between. The thresholds for obtaining upper edges are $[-0.050, 0.050]$ for x and $[0.020, 0.050]$ for y, and the thresholds for obtaining lower edges are $[-0.050, 0.050]$ for x and $[-0.100, 0.013]$ for y. Notice that the thresholds for x for both cases are the same, this is due to the hole directions being horizontal so that only the gradient in x direction is enough for the detection. For a more complicated hole or damage area, gradients in both x and y are needed for the detection of edges. Also notice that some noise is found on the left side of the slab, as shown in Fig. 3.14 (a). This is mainly due to the heterogeneity of concrete, and also uneven heating by the thermal blanket. The comparison of detected region and actual holes is shown in Fig. 3.14 (b), and visual comparison shows good agreement; a more quantitative comparison is discussed below.



(a)



(b)

Figure 3.14 Image after Sobel filtering (a) holes detection based on the upper and lower edges (b) comparison between detected holes and ground truth; blue: detected holes, green: ground truth

3.3.5.5 Performance Discussion

Now we discuss the hole detection performance for different sample rates. In order to evaluate the performance quantitatively, a score is defined as the ratio of correctly detected area to the total detected area. As the sampling rate increases, the score grows accordingly (Fig. 3.15). The score increases by almost 40% (i.e., $100\% \times (0.723 - 0.523)/0.523$, as the sampling interval decreases from 2 mins to 1 second. This indicates that by increasing the sample rate, the damage detection performance can be greatly improved. However, this increases the demand on the data analytics computation, which is resolved by the MapReduce technique.

Compared with the traditional single machine computation, the computational expense (time cost) is greatly reduced as shown in Table 3.1. Notice that via distributed computation, the time cost is only 10 of local computation. It can be seen that as the number of nodes being used increases, the corresponding speedup increases almost linearly, which illustrates the scalability of MapReduce. Also notice that as the number of nodes increases, the computational time decays similar to exponential decay (Fig. 3.16).

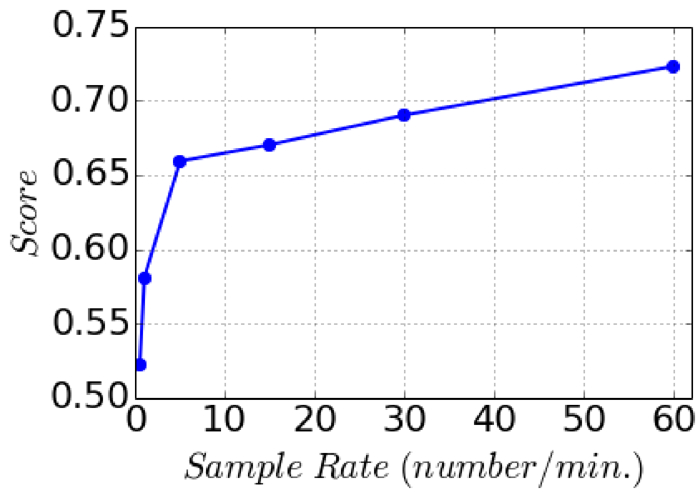


Figure 3.15 Detection performance vs. sampling rate

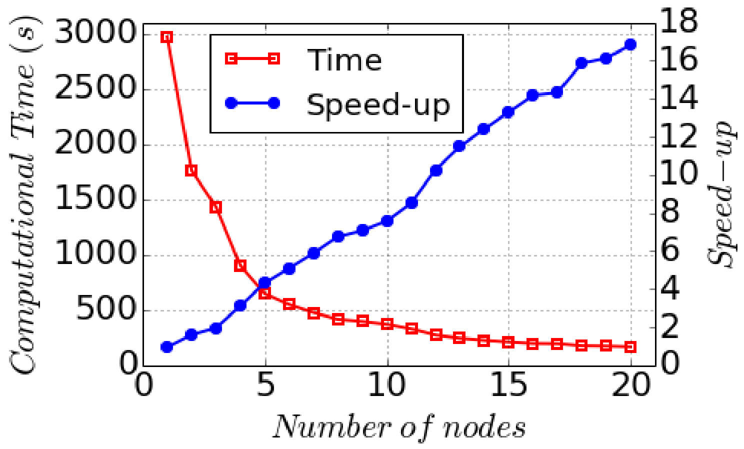


Figure 3.16 Thermography camera and the specimen to be monitored

However, the time spent by the traditional method is $1560\ s$, while the MapReduce method on a single node takes as much as $2971\ s$. This is due to two reasons. First, the operations related to MapReduce such as data transferring, data splitting, task managing, and mapping cost additional time. Second, the CPU and memory of the cluster node is less powerful (in this example) than the computer client used for local traditional computation (Table 3.2).

Table 3.1 Time cost of traditional method and MapReduce method

Method	**Time (s)**
Traditional	1560
MapReduce (20 nodes)	163

Table 3.2 Node used by traditional method and MapReduce method

Method	**CPU (GHZ)**	**Memory (GB)**
Traditional	$3.4\times\ 8$	12
MapReduce (20 nodes)	2.3	5

The time cost of individual step in data processing (for one image) is shown in Table 3.3. For this simple case, data reading accounts for a large portion of the total time. However, for more complicated data processing, actual processing is expected to occupy a much larger portion.

Table 3.3 Time cost of individual steps in data processing

Step	Time (s)
Data Reading	0.14
Cropping	0.08
Uniform filtering	0.08
Sobel filtering	0.07

3.4 Summary

This chapter developed a framework for applying a big data technique to structural health monitoring, in particular image processing. The popular MapReduce approach was applied in the proposed framework, and realized via Apache Spark. Structural damage detection was parallelized via MapReduce, by transforming inputs and outputs as key-value pairs. Sobel filter was used for illustration of the image processing. It can be easily replaced with other appropriate techniques for different scenarios. Results show that the processing effort scaled well, in an almost linear trend. The approach was illustrated for the processing of thermal images obtained for a concrete slab, and the data volume is less than $20\ GB$. For practical structural health monitoring for the whole structure in the field, the data can be very large, thus considerably increasing the advantage of MapReduce in realistic application.

Note that this chapter only considered the application of big data techniques to deterministic structural health monitoring; extension to uncertainty quantification in diagnosis will be considered in future chapters. Second, this chapter did not consider the complexity problem of parallelization in MapReduce, which can lead to different parallelization options via splitting the task data-wise or function-wise. Third, fault-tolerance is an important issue in big data analytics, which needs to be incorporated in future work.

CHAPTER 4

UNCERTAINTY QUANTIFICATION IN DIAGNOSIS AND PROGNOSIS

4.1 Background

Two common problems encountered by engineers are prediction of system response to different input conditions (in order to support decisions regarding system design, operational conditions, and risk management activities such as inspection, maintenance and repair), and inference of system state or system model parameters given observations regarding one or more response variables. Prediction is a forward problem, and inference is an inverse problem. Both types of problems are affected by many different sources of uncertainty, which may be classified into two types: aleatory and epistemic. Aleatory uncertainty refers to natural variability, which is irreducible (e.g. material parameters). On the other hand, epistemic uncertainty is due to lack of knowledge, which could be reduced when new information becomes available. Examples of epistemic uncertainty are information uncertainty regarding the model inputs or model parameters (due to inadequate or imprecise data) and model uncertainty (due to assumptions and approximations in modeling the reality). Model errors, which include numerical solution errors and model form errors, can be quantified through calibration, verification and validation activities and included in the reliability analysis. Structural health monitoring consists of both the forward and inverse problems, namely diagnosis (inverse problem) and prognosis (forward problem), both of which are affected by aleatory and epistemic uncertainty sources. It is necessary to identify the uncertainty sources and quantify their effects on diagnosis and prognosis, in order to facilitate effective risk management. This chapter investigates efficient computational approaches for uncertainty quantification in both forward and inverse problems, and illustrates them for structural health monitoring.

This chapter focuses on the following issues: 1. Investigation of techniques to parallelize the Bayesian inference for diagnosis uncertainty. Popular numerical techniques for Bayesian inference, namely Markov chain Monte Carlo (MCMC) and particle filter (PF) will be parallelized, including strategies for fault tolerance. 2. Investigation of big data techniques for efficient quantification of uncertainty in damage prognosis. The repeated FEA model runs in Monte Carlo simulation will be parallelized to reduce the computational cost of uncertainty propagation analysis. The prognosis objective is to quantify the probability distribution of predicted damage growth and remaining useful life (RUL) (Farrar and Worden, 2007) of the structure.

This chapter utilizes big data techniques to analyze voluminous SHM data (i.e., image files) for damage diagnosis, and to quantify the diagnosis uncertainty. Prognosis is realized using a damage growth model coupled with FEA, and the remaining useful life (RUL) is estimated. The uncertainty in the diagnosis of the structural state is then propagated to the prognosis result, in addition to uncertainty sources in the structural properties, usage and environment. The use of big data analysis techniques makes uncertainty quantification feasible in terms of computational effort, by efficiently quantifying and aggregating the uncertainty from multiple sources.

Note that this chapter focuses on the MapReduce application of handling uncertainty quantification in diagnosis and prognosis. The application of MapReduce to SHM data processing (deterministic diagnosis) was already discussed in Chapter 3. The details of MapReduce implementation for data processing have been explained in Sec. 3.3. The basic steps in implementing MapReduce for image or signal processing in structural health monitoring, as discussed in Chapter 3, are: (1) upload the acquired data from the local computer to the cluster of computers; (2) prepare the data processing functions, and substitute into the Map function shown in Pseudocode 3.1 (Fig. 3.5); and (3) run Spark to process and retrieve the data files from the cluster back to the local computer.

4.2 MapReduce for Diagnosis under Uncertainty

Since damage diagnosis under uncertainty is pursued using Bayesian methods in this chapter, we first describe the general steps of parallelizing Bayesian updating methods. For sample based Bayesian updating methods, the posterior distribution is approximated by samples, which is gradually available. The main idea is to split the sampling tasks to cluster nodes, and estimate the posterior after all tasks completed and with samples transferred to the master node (Fig. 4.1). The main steps in the parallelization of Bayesian updating methods are summarized as below:

Step 1. Set the parameters (number of samples, burn-in length etc.).

Step 2. Use MapReduce to assign the sampling task to cluster nodes.

Step 3. Re-assemble the samples and construct the posterior distribution.

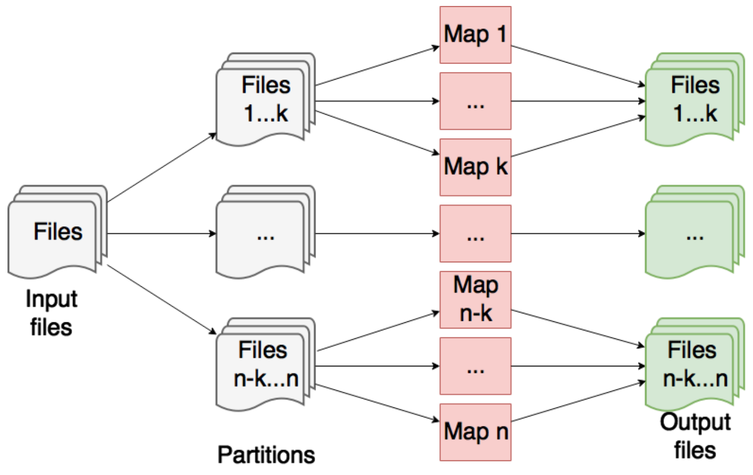


Figure 4.1 Schematic description of the MapReduce process

This basic approach is applied to two sampling-based Bayesian methods below, namely Markov Chain Monte Carlo (MCMC) sampling, and Particle Filter (PF).

4.3.1 MapReduce for Markov Chain Monte Carlo

The MCMC method was described in Chapter 2. The basic idea of MCMC parallelization is to divide the observations into M splits, with each node taking one partition to provide samples of the posterior distribution. The prior distribution of the variable of interest will be updated using the equation (Neiswanger et. al, 2013):

$$p_m(\theta) \propto p(\theta)^{\frac{1}{M}} p(x^{n_m}|\theta) \tag{4.1}$$

After all nodes complete their tasks, all the sub-posterior samples from each nodes will be combined to produce samples for an estimate of the sub-posterior density product $p_1, \cdots, p_M$, which is proportional to the full data posterior, i.e. $p_1, \cdots, p_M(\theta) \propto p(\theta|x^N)$. Pseudocode 4.1 in Fig. 4.2 shows the implementation of MCMC use MapReduce.

```
Pseudocode 4.1:

function ParameterSetting()

mapper(x):
   function ReadData()
   function MCMC_Sampling()
   function SaveSamples()
   return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()

function PosteriorEstimate()
```

Figure 4.2 MapReduce implementation of MCMC

A Map function is defined (’mapper’), within which all the actual functions are defined (Read Data(),MCMC_Sampling(), and SaveSamples()). As shown in Fig. 4.2, the sampling process is executed on the slave nodes, while posterior integration is done after all particles and weights are saved from the slave nodes. SparkContext and count() function are used the same way as in Pseudocode 3.1 in Fig 3.5. ReadData() is the function used to read observation data and parameters, and followed by MCMC_Sampling(), which is the function to perform the sampling. SaveSamples() is the function used to save all subset of MCMC chains. After all samples are saved, the function PosteriorEstimate() will be called to construct the posterior distribution based on samples.

4.3.2 MapReduce for Particle Filter

The particle filter method was described in Chapter 2. In order to reduce the computational cost, particle filter is parallelized in this study using MapReduce, which is implemented in Spark. Pseudocode 4.2 in Fig. 4.3 summarizes this approach.

```
Pseudocode 4.2:

function ParameterSetting();

mapper(x):
    function ReadData()
    function Sampling()
    function SaveParticles()
    function SaveWeights()
return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()

function ReadData()
function Sampling()
function SaveParticles()
function SaveWeights()
function PosteriorEstimate()
```

Figure 4.3 MapReduce implementation of Particle Filter

Similar to the MapReduce application of data processing, a Map function is defined (‘mapper’), within which all the actual functions are defined (reading, sampling, and saving). SparkContext and count() function are used the same way as in Pseudocode 3.1. As shown in Pseudocode 4.2, the sampling process is executed on the slave nodes, while resampling is done after all particles and weights are saved from the slave nodes. ReadData() is the function used to read observation data and parameters, and followed by Sampling(), which is the function to perform the sampling. Note that ReadData() occurs both inside and outside the mapper function, which means that data reading happens both on slave nodes and the master node. By doing this, there is no direct data transfer between nodes, which further saves computational time, and avoids faults that might happen during the communication (such as loss of data and miscommunication). After particles and weights are saved, the posterior distribution can be approximated by function PosteriorEstimate().

4.4 MapReduce for Prognosis Uncertainty Quantification

Damage prognosis needs to propagate uncertainties, which applies Monte Carlo sampling to repeatedly run FEA simulations and damage growth models. MapReduce can be used to parallelize those runs efficiently.

```
Pseudocode 4.3:

function ParameterSetting()

mapper(x):
   function InputData = ReadData()
   function OutputData = FEA_Processing()
   function WriteData(OutputData)
return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()
```

Figure 4.4 MapReduce implementation of MCS

Since MCS needs repeated FEA runs with different inputs, parallelization can be realized by using MapReduce. Fig. 4.4 shows the implementation in Spark. Similar to the MapReduce application in data processing, a Map function is defined (‘mapper’), within which all the actual functions are defined (reading, processing, and saving). ReadData() is the function used to read FEA configurations (realizations of control variables from Monte Carlo Simulation), followed by FEAProcessing(), which is the function to perform the sampling.

In summary, to reduce the computational effort in uncertainty quantification of structural diagnosis and prognosis in the context of big data, we proposed the methodology of parallelization of SHM data processing, diagnosis UQ and prognosis UQ. Note that the MapReduce procedure can be easily extended to the general inverse and forward problems encountered in uncertainty quantification analyses, although it is explored here within the context of structural health monitoring.

4.5 Numerical Example: ASR Diagnosis and Prognosis in Concrete

4.5.1 Background of ASR Degradation in Concrete

Alkali-silica reaction is a reaction between the alkali in the cement and reactive silica in the aggregate in concrete structures. The reaction product is a gel which expands in the presence of moisture, eventually causing cracking. The chemical reaction can be described in two steps: alkali-silica gel formation and alkali-silica gel expansion (Saouma and Perotti, 2006). The gel formation can be represented using the chemical equation below:

$$[xSiO_2] + [yNa(K)OH] \rightarrow [Na(K)_ySi_xO_zaq] \tag{4.2}$$

And the expansion of the alkali-silica gel in the presence of moisture is represented as:

$$\left[Na(K)_ySi_xO_zaq\right] + [H_2O] \rightarrow [Na(K)_ySi_xO_zH_2O] \tag{4.3}$$

The expansive stress results in micro- to macro- cracking. The cracking increases the permeability of the concrete, causing increased moisture ingress and therefore further gel expansion and cracking.

4.5.1 ASR Description and Modeling

Saouma and Perotti (2006) presented a comprehensive coupled thermo-hydro-mechanical chemical (THMC) model for ASR gel expansion based on Ulm et al. (2000), and considered the effects of stress on the reaction kinetics and anisotropic volumetric expansion induced by ASR. We applied this model using the Abaqus FEA software, by programming the constitutive model in a user-defined material (UMAT) code. By choosing the appropriate parameters, this model can simulate ASR expansion in a realistic manner, based on several advanced features: 1. ASR expansion strain is treated as a full strain tensor, not calculated separately and independently for each principal direction; 2. ASR reaction rate is temperature dependent; 3. ASR reaction can be retarded by compressive stress within concrete; 4. ASR expansion is constrained by compression, and is redirected into other less-constrained principal directions; 5. both high compressive or tensile stress states inhibit ASR expansion due to the formation of micro- and macro-cracks that absorb the expanding gel; 6. triaxial compressive stress state reduces expansion; and 7. reduction in tensile strength and elastic modulus are included in the model.

4.5.1.1 ASR Reaction Kinetics

Based on Ulm et. al (2000)'s stress-independent reaction model, Saouma and Perotti (2006) proposed a first order ASR reaction kinetics model that is dependent on both the temperature and the first invariant of the stress tensor as:

$$t_C(\theta,\xi)\cdot\frac{d\xi}{dt} = \tau_C(\theta)\cdot\frac{1+\exp\left[-\frac{\tau_L(\theta,I_\sigma,f_c')}{\tau_C(\theta)}\right]}{\xi+\exp\left[-\frac{\tau_L(\theta,I_\sigma,f_c')}{\tau_C(\theta)}\right]}\cdot\frac{d\xi}{dt} = 1-\xi \qquad (4.4)$$

in which ξ is the ASR reaction extent ranging from 0 (not reacted) to 1 (fully reacted); θ is the temperature; τ_C is characteristic time constant, while τ_L is latency time constant; I_θ is the first invariant of the stress tensor; f_c' is the uniaxial compressive strength of concrete. See (Ulm et. al, 2000) for detailed discussion of these variables.

4.5.1.2 Stress-dependent ASR Volumetric Strain

Once the increment of ASR reaction extent $\Delta\xi$ is obtained, the ASR volumetric strain increment $\Delta\epsilon_{vol}^{ASR}$ can be evaluated as:

$$\Delta\epsilon_{vol}^{ASR} = \Gamma_t(f_t',\sigma_I|COD)\Gamma_c(\bar{\sigma},f_c')g(H)\Delta\xi\epsilon^{\infty}|\theta=\theta_0 \qquad (4.5)$$

where f_t' is the tensile strength of the concrete; I is the maximum principal stress (>0 under tensile stress); COD is the crack opening displacement; $\bar{\sigma}$ is the ratio between the hydrostatic stress and compressive strength of concrete, and ϵ^{∞} is the laboratory-determined maximum free volumetric expansion at the reference temperature θ_0. Γ_t accounts for ASR reduction due to tensile cracking, while Γ_c accounts for the reduction in ASR volumetric expansion under compressive stresses (in which case gel is absorbed by diffused microcracks). See Saouma and Perotti (2006) for detailed discussion of the above variables and functions.

4.5.1.3 Anisotropic ASR Strains and Weights in Principal Directions

The incremental ASR volumetric strain Δ_{vol}^{ASR} needs to be redistributed along three principal directions according to their relative propensity to expand. Saouma and Perotti (2006) presented a method to calculate the relative weights along the three principal directions based on the principal stresses under either uniaxial, biaxial or triaxial confinement conditions. Given the full stress tensor (in Cartesian coordinates) on a quadrature point within an element, an eigen-solver is used to obtain the three principal stresses, k, l and m, and associated eigen-vectors along the directions of principal stresses, R_k, R_l and R_m. These eigen-vectors form a stress/strain rotational matrix $R = R(R_k, R_l, R_m)$ that will be used later to rotate the incremental ASR strain tensor expressed in principal stress/strain coordinates back into Cartesian coordinates. ASR expansion weights W_k, W_l, W_m along the principal directions can be obtained following the procedure described in Saouma and Perotti (2006), given concrete tensile strength f_t', compressive strength f_c', and a gel expansion inhibiting compressive strength σ_u. After obtaining the weights, the individual incremental ASR strains along the principal directions are then obtained using these weights by the following formula

$$\Delta\epsilon_i^{ASR} = W_i \Delta\epsilon_v^{ASR}, i = 1, 2, 3 \tag{4.5}$$

Finally the full ASR expansion-induced incremental strain tensor $\Delta\epsilon^{\mathrm{ASR}}$ can be obtained by rotating $\Delta\epsilon_i^{ASR}$ on quadrature points via

$$\Delta\epsilon^{ASR} = R\Delta\epsilon_i^{ASR} R^T \tag{4.6}$$

4.5.1.4 Reduction of Elastic Modulus and Tensile Strength

The ASR-induced deterioration of concrete mechanical properties is simply modeled as a time-dependent function of ASR reaction extent $\Delta\xi(t, \theta)$ following Saouma and Perotti

(2006):

$$E(t,\theta) = E_0[1-(1-\beta E)\xi(t,\theta)] \tag{4.7}$$

$$f_t(t,\theta) = f_{t,0}[1-(1-\beta E)\xi(t,\theta)] \tag{4.8}$$

where E_0 and $f_{t,0}$ are the original elastic modulus and tensile strength, respectively; and β_E and β_f are the corresponding residual fractional values when the concrete has fully reacted. Both β_E and β_f are input parameters chosen by user.

4.5.2 Experiment

The objective in this example is to diagnose the ASR damage in a cement slab which is cast and cured in the laboratory, and to predict future damage. Using sodium hydroxide, $NaOH$, in the mix water or placing the cured concrete in a $NaOH$ solution causes an increase in pH, thus accelerating the chemical reaction and ASR gel formation. Glass slides are placed inside the cement slab ($C1$ and $C2$) to provide the silica for the reaction. For the purpose of baseline removal, another set of specimens are cured in H_2O ($A1$ and $C1$). For each group, a specimen without glass ($A1$ and $A2$) is also prepared to serve as the control group. The specimen configurations are shown in Table 4.1. The dimensions of the slabs are $5\ in \times 9\ in \times 2\ in$.

The mechanics of damage detection using infrared thermography is based on the differences in heat transfer properties of different materials. The ASR gel in the structure has a lower thermal conductivity coefficient than cement, which will lead to a ‘lagging‘ phenomenon, i.e., the heating and cooling time of the gel are slower than the surrounding cement. The slab is placed on a HEATCON ® thermal blanket and uniformly heated from below. Each thermal cycle has a total duration of 70 minutes. The heating profile is shown in Fig. 3.9; the temperature values are scaled to the range $(0, 1)$ due to export control reasons. The camera was setup to capture images of the concrete slab every 0.5 minute.

Table 4.1 Configuration of specimens

Specimen	Solution	Glass
$A1$	H_2O	No
$C1$	H_2O	Yes
$A2$	$NaOH$	No
$C2$	$NaOH$	Yes

4.5.3 Uncertainty Sources in Diagnosis and Prognosis

First, let us consider the aleatory and epistemic sources specific to ASR diagnosis and prognosis. For diagnosis, specimen variability (e.g. specimen dimensions and material properties) is aleatory uncertainty, when considering variation across multiple specimens. However, for a single specimen, these quantities are unique, and the uncertainty related to them is epistemic, i.e., not knowing their actual values. Measurement error (from sensors) is aleatory uncertainty. However, the data processing steps (e.g., cropping, filtering, smoothing, feature selection etc.) incorporate several assumptions and parameter selections by the analyst, which will cause epistemic uncertainty. Assumptions in Bayesian updating (prior distribution) as well as the choice of tuning parameters in numerical algorithms such as MCMC and PF create epistemic uncertainty. In prognosis, aleatory uncertainty is introduced by loading loading variability. On the other hand, epistemic uncertainty is propagated from diagnosis uncertainty, in addition to model errors in FEA (e.g., discretization error) and the ASR expansion model (model form error). In this example, we only considered diagnosis uncertainty, which is caused by measurement error. Since we are performing diagnosis and prognosis for this single specimen, there is no aleatory uncertainty regarding its properties (i.e., no variability across multiple specimens since we are only considering a single specimen).

4.5.4 Data Processing

Damage in concrete due to alkali-silica reaction is detected through image processing of infrared thermal images. In this application, image processing is simply a subtraction between the image of the control specimen (healthy structure) and the image of the test specimen (damaged structure). Since multiple images are obtained for a single test, the image pair that has the largest difference is chosen. Then by setting an appropriate threshold for the temperature difference, the magnitude of area under ASR damage can be estimated. The implementation of various steps in processing the thermal image data are discussed in detail and the results are presented below.

4.5.4.1 Cropping

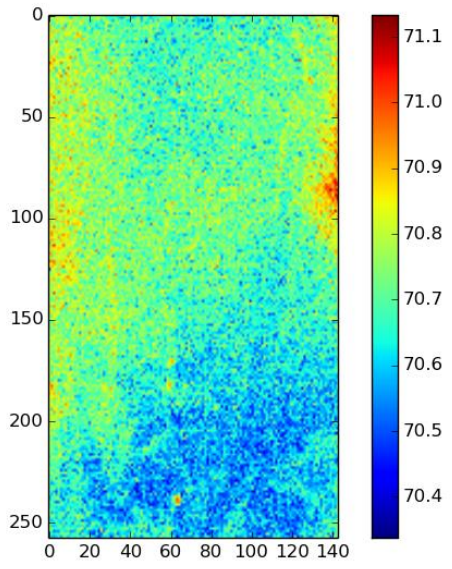
The raw image needs to be cropped in order to achieve greater resolution in analyzing the temperature distribution within the slab. After several trials, the appropriate pixel range for cropping was found to be $[123\!:381, 443\!:586]$ for $A1$, $[132\!:390, 47, 190]$ for $C1$, $[130 : 388, 427 : 570]$ for $A2$, and $[138\!:396, 28\!:171]$ for $C2$. The cropped images are shown in Fig. 4.5. For each image, the resolution is 258×143 pixels.

4.5.4.2 Baseline Removal

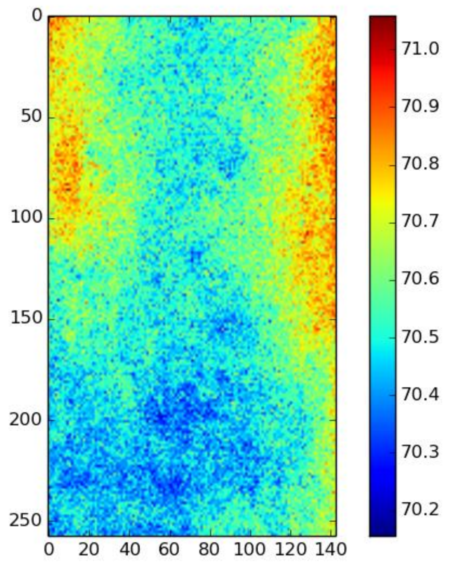
Specimens $A1$ and $C1$ are cured in H_2O, while $A2$ and $C2$ are cured in $NaOH$. Baseline removal is realized by subtracting the cropped thermal image $A2$ from $A1$, and $C2$ from $C1$. The images after baseline removal are shown in Fig. 4.6. This is based on the hypothesis that the formation of ASR should change the heat conductivity within the slab. Therefore, temperature difference between the H_2O-cured and $NaOH$-cured slabs at each time point is expected.

4.5.4.3 Feature Extraction

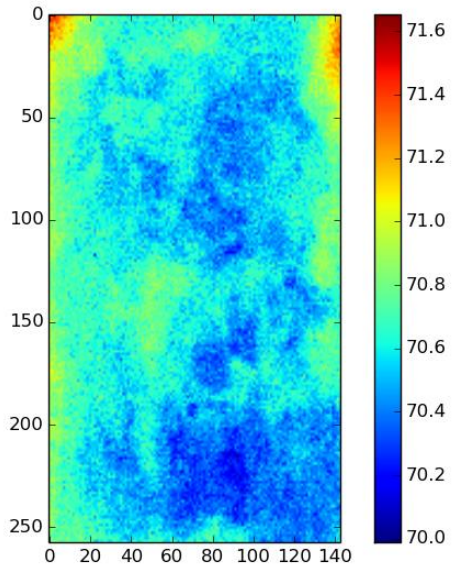
Based on the baseline slab A, we selected upper bound and lower bound values (at each time instant) for the temperature difference between H_2O-cured data and $NaOH$-cured data. If the temperature difference between slab $C1$ and $C2$ is outside the bounds we treat it as indicating a change in heat conductivity, thus implying the formation of ASR. Otherwise, we treat it as normal, i.e., no ASR has formed. To set boundaries, we selected the maximum and minimum values of the temperature difference among all pixels between the H_2O-cured data and $NaOH$-cured data for slab A, at each time point. Fig. 4.7 shows one example of ASR damaged region. Seven inspections (with time interval of 10 days) are obtained, which is plotted in Fig. 4.8.



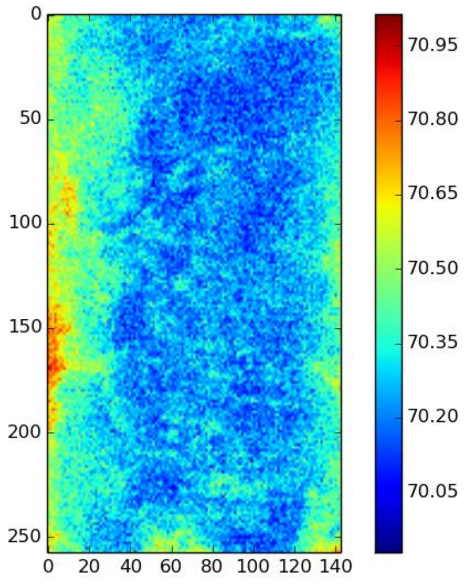
(a)



(b)

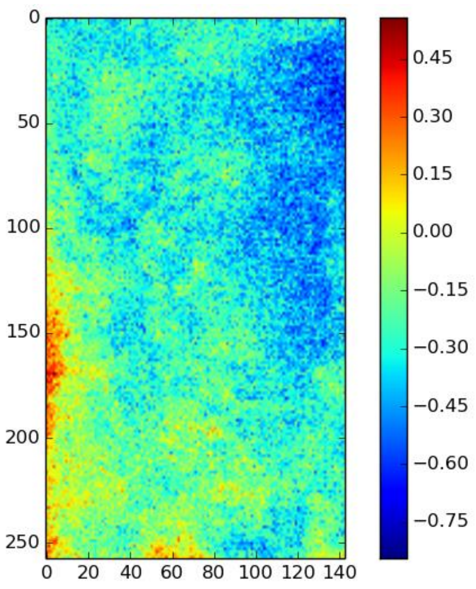


(c)

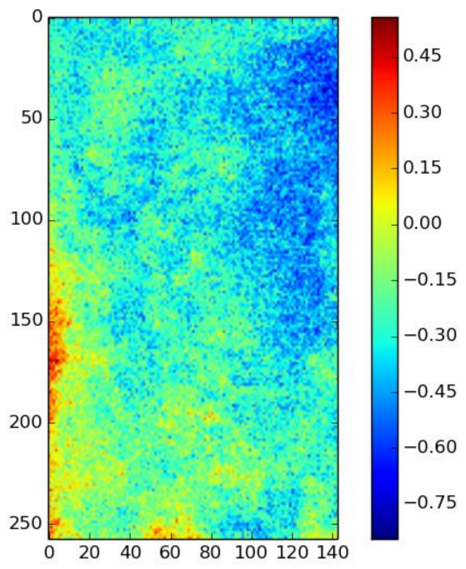


(d)

Figure 4.5 Cropped images (a) specimen $A1$; (b) specimen $C1$; (c) specimen $A2$; (d) specimen $C2$

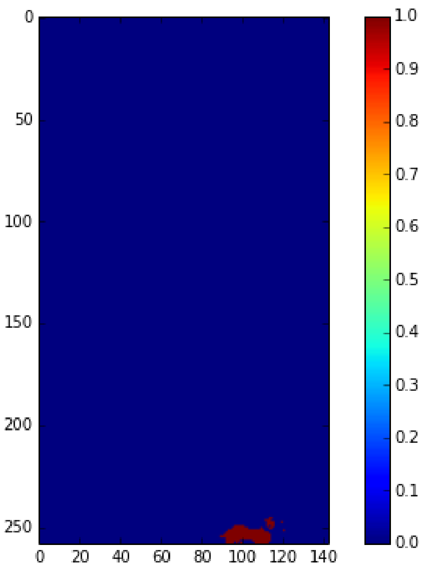


(a)

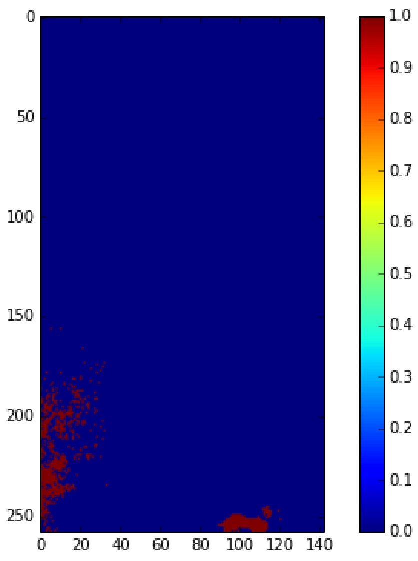


(b)

Figure 4.6 Images after baseline removal (a) specimen A; (b) specimen C



(a)



(b)

Figure 4.7 ASR damaged region after feature extraction. Red: ASR damage; Blue: healthy concrete. (a): Inspection 1 (t = 30 days); (b): Inspection 2 (t = 40 days)

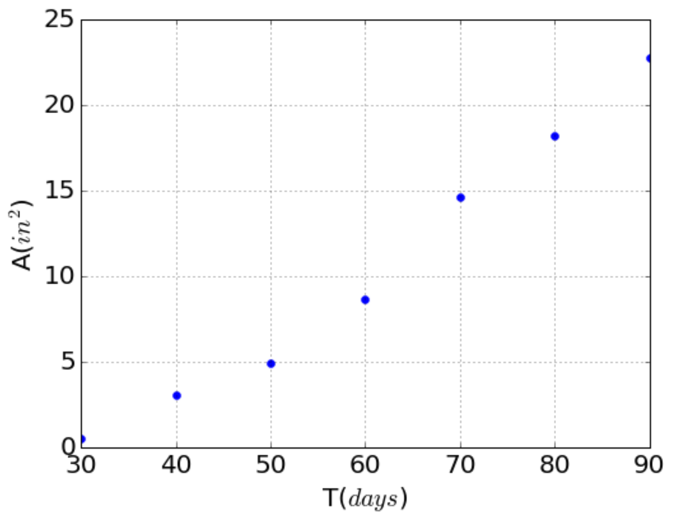


Figure 4.8 ASR damaged area at different inspection time points

4.5.5 Diagnosis

For each inspection point, Bayesian updating is used to obtain the posterior distribution of the true ASR area based on the detected ASR area value. The Particle Filter method implemented in MapReduce (as described in the previous section) is used to perform this computation. A non-informative uniform prior ($\sim Uniform(0, 20)$) is assumed for ASR damaged area A, and a normal distribution ($\sim N(0, \sigma)$) is used to represent the measurement error, where a uniform prior ($\sim Uniform(0.1, 1)$) is assumed for σ. The posterior distribution is shown in Fig. 4.9. In this example, $50,000$ particles for PF and $50,000$ samples for MCMC were used.

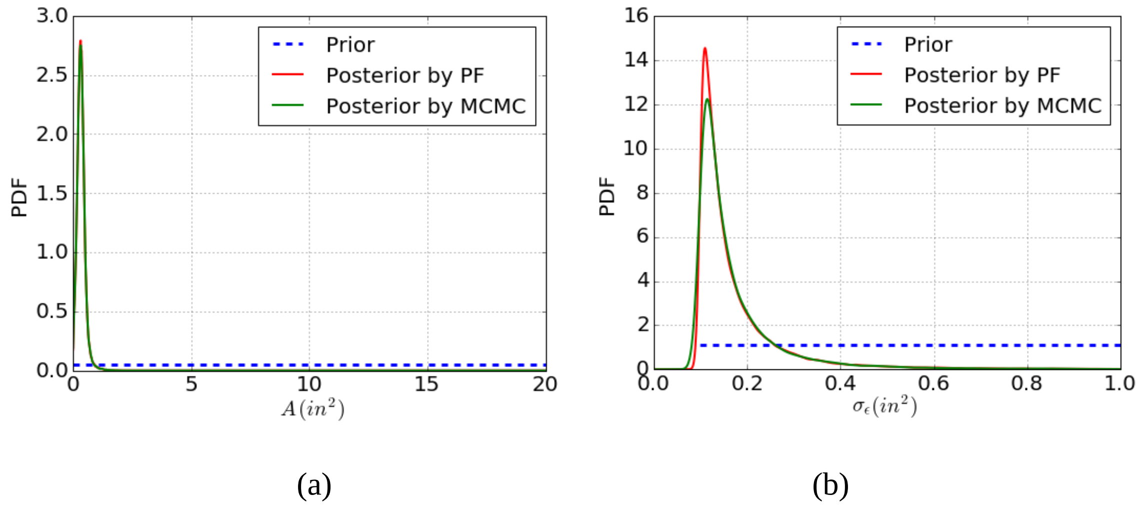


Figure 4.9 Bayesian updating (@ T = 30 days) for (a): ASR damaged area *A* and (b): observation error standard deviation *σ*

In diagnosis, 20 cluster nodes were used for parallelization. For the purpose of comparison, computation using the traditional method (single processor) was also performed. The computational power of desktop and cluster nodes are compared in Table 4.2. It is worth noting that the CPU clock speed and memory size of the local machine where the traditional methods were running are larger than that of the cluster nodes. The comparison between the time cost of traditional method and MapReduce method for this study is shown in Table 4.3. In both PF and MCMC Bayesian updating, MapReduce does not show a significant advantage. This is mainly due to two reasons. First, the power of the cluster node is lower than the local computer. Second, which is more important, the computational cost for each split of PF and MCMC chain is low, which led to dominance of the communication time between master node and slave nodes. The MapReduce method will show its advantage as the observation data size becomes larger and when the problem is high-dimensional, i.e., when the PF and MCMC sampling demands are larger than the communication demands.

Table 4.2 Node comparison

Method	**CPU (GHZ)**	**Memory (GB)**
Desktop	3.4×8	12
Cluster Nodes	2.3	5

Table 4.3 Time cost comparison for Bayesian updating

Method	**PF (s)**	**MCMC (s)**
Traditional	3.2	2.4
20 cluster nodes	4.5	4.1

4.5.5 Prognosis

To predict the ASR damaged area growth, two steps are needed. First, the current ASR damaged area is sampled from the posterior distribution obtained by Bayesian updating, to account for the uncertainty in the diagnosis. Since the Bayesian updating is performed using Particle Filter, the posterior samples generated by the Particle Filter can be directly used, instead of constructing an approximate posterior distribution (typically done using kernel density functions) and then sampling from that. Second, ASR gel expansion model (implemented in combination with FEA analysis) is utilized to predict the growth of ASR.

4.5.5.1 ASR Gel Expansion Modeling

We implemented Saouma and Perotti's ASR gel expansion model (Saouma and Perotti, 2006) using Abaqus. The ASR region identified from the previous diagnosis is considered as the initial condition in the FEA model. To be realistic, isotropic expansion or shrinkage will be made based on the original detected damaged area from image processing. For example, if the diagnosed damaged area from Bayesian updating is greater on the detected area from image processing, the outer surroundings of the current area will be considered as damaged also. When the number of elements to be added cannot occupy the whole surrounding layer, part of the surrounding layer will be chosen randomly. It is the similar case when the diagnosed damaged area by Bayesian updating is smaller than the detected area from image processing techniques. This can guarantee that the adjusted area is closest to the diagnosed area from Bayesian updating. Temperature, humidity and mechanical constraints are considered as boundary conditions. By running the FEA model, the future status of the ASR damage area is predicted. Note that the diagnosed ASR region from image processing is represented by pixels. However, the structure is represented using elements in FEA. Therefore an approximation was made to convert the ASR detection result to the FEA model. In detail, the element is considered to be occupied by ASR gel ($\xi = 1$) if more than half of the pixels within it are positive in detection. Furthermore, to be more realistic, a linear function is defined at the boundary of the ASR region to allow a gradual decrease in ASR reactive extent. Fig. 4.10 (a) shows an example of the initial condition of the FEA model.

Table 4.4 Parameters of the ASR model

Characteristics	Symbol	Unit	Value
Maximum volumetric ASR strain at test temperature T_0^{test}	ε^{∞}	$-$	0.00262
Characteristic time at test temperature T_0^{test}	τ_C	day	τ_C
Latency time at test temperature T_0^{test}	τ_L	day	110.0
Activation energy associated with τ_C	U_C	K	$5,400$
Activation energy associated with τ_L	U_L	K	$9,400$
Residual reduction factor	Γ_r	$-$	0.5
Tensile strength	$f_t^{'}$	MPa	3.2
Residual reduction factor for ASR expansion under tensile stress	γ_r	$-$	0.5
Fraction of $f_t^{'}$ prior to reduction of ASR expansion due to macro-cracking	γ_t	$-$	0.5
Compressive strength	$f_c^{'}$	MPa	-31
Upper compressive stress beyond which there is no more ASR expansion	σ_u	MPa	-8
Concrete Young's modulus	E_0	GPa	37.3
Concrete Poisson's ratio	ν	$-$	0.22
Reduction fraction for Young's Modulus at end of ASR reaction	β_E	$-$	0.5
Reduction fraction for tensile strength at end of ASR reaction	β_f	$-$	0.5

Each sample of the diagnosed ASR damaged area posterior is treated as an individual initial condition to the FEA model. The room temperature ($298.15°K$), humidity (40%) and free boundary are considered as boundary conditions. Fig. 4.10 (a) gives an example of how the diagnosis result of ASR is incorporated within the FEA model at any time step. The parameters used in this study are listed in Table 4.4. Note that τ_C is treated as an unknown model parameter, which needs to be calibrated in each inspection step. Eq. (21b) in Ulm et al. (2000) shows the effect of τ_C in ASR development.

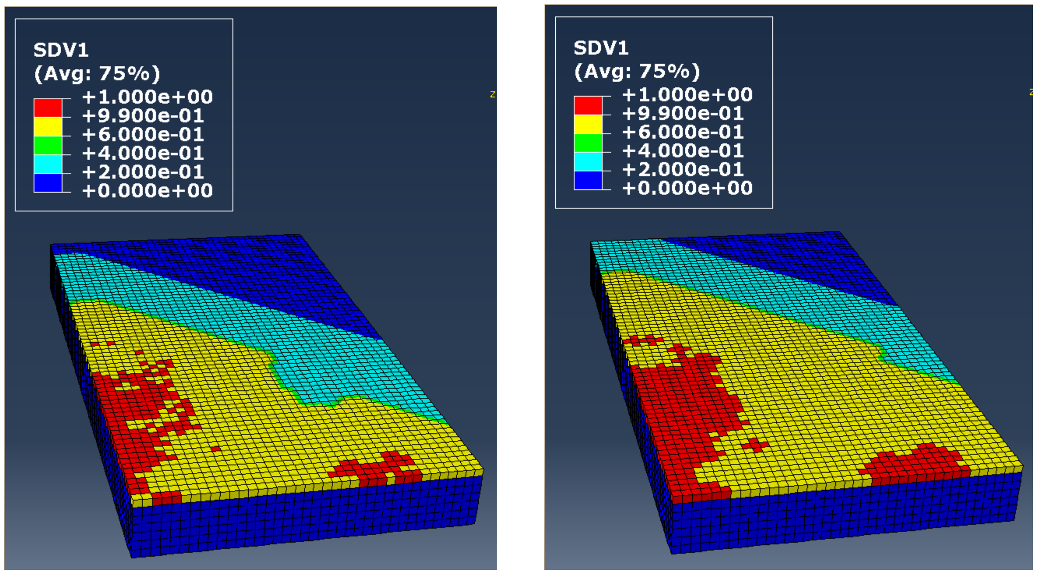


Figure 4.10 FEA model input and output (half model)

4.5.5.2 ASR Damaged Area Prognosis

Fig. 4.10 (b) gives an example of FEA model prediction of ASR growth starting from the diagnosis in Fig. 4.10 (a). Note that the prediction can only expand the ASR area, but not add new ASR affected regions that are not connected to the input area; however, new inspection data may indicate new unconnected damaged regions and can be incorporated in the FEA model for subsequent predictions. Each element will be considered as fully occupied by ASR when the ASR extent (ξ) is greater than 0.99. Thus the ASR damaged area can be predicted by the model. The ASR damaged area prognosis with 95% probability bounds is shown in Fig. 4.11. The 95% bounds are formed based on Monte Carlo samples from the Bayesian updating posterior. As shown in Fig. 4.11, a generally increasing trend is found for the ASR damaged area. The inspection data is given every 10 days (marked with arrows). Note that the prediction variance (as indicated by the 95% prediction bounds) increases from the beginning to the end of each time period (10 days) as expected, and decreases at inspection since the area has been measured. Thus the prognosis for each time period starts from the measured area, and the variance at the beginning of each time period is only due to measurement error. It is also worth noting that the variance at the end of each time period reduces as we move from one time period to next, thus indicating reduction in model uncertainty over multiple inspections.

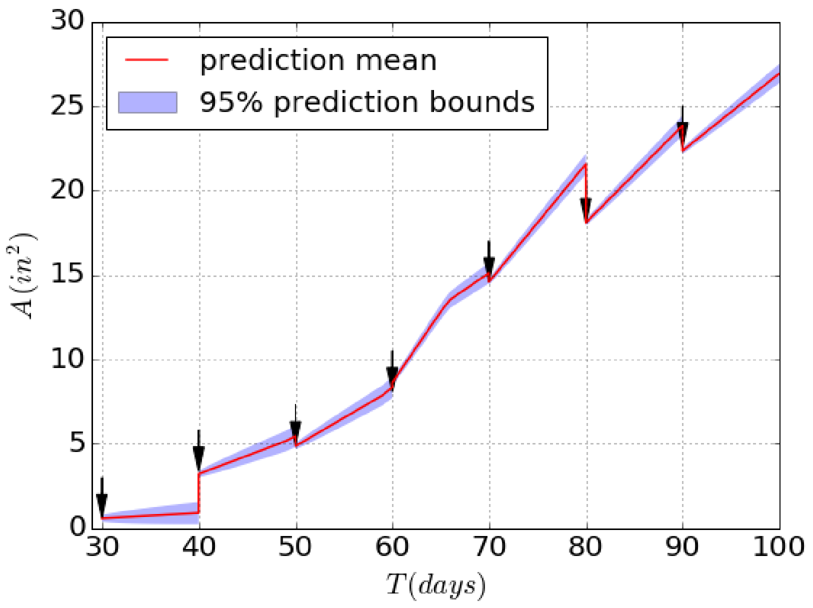


Figure 4.11 ASR damaged area prognosis and uncertainty quantification

The computational power of the desktop (single node) and cluster nodes are compared in Table 4.5. Abaqus FEA runs were parallelized using 5 desktop cores (due to limited number of available licenses), which reduced the computational cost down to around one fifth. Compared to the MapReduce for Bayesian updating (Table 4.5), the efficiency is greater due to much smaller communication time spent when performing parallelization locally.

Table 4.5 Time cost comparison for prognosis

Method	**MCS (s)**
Traditional	1506.7
5 desktop cores	307.2

4.5.5.3 Remaining Useful Life

The threshold for ASR damaged area is assumed as $A_{th} = 27\ in^2$, beyond which the structure will be considered as failure. The remaining useful life prediction with 95% bounds is shown in Fig. 4.12. A decreasing trend of the RUL along time is observed. The corresponding failure probability is shown in Fig. 4.13. Until $T = 98$ days, the failure probability is almost zero, since the threshold damage ($27\ in^2$) is several standard deviations away from the mean prediction. (This is also seen from Fig. 4.12, where the RUL is far away from zero). At $T = 100$ days, the inspection indicates a higher probability of failure, which is consistent with ASR damaged area in Fig. 4.11 and RUL in Fig. 4.12. Note that the variance within each time period (10 days) is constant because the prediction of RUL is only at the beginning of the time period (thus there is only one value of variance), but in the plot, the RUL is continuously reduced by the number of days within each time period. It is worth noting that the variance in the RUL prediction decreases over multiple time periods, indicating reduction in model uncertainty over time.

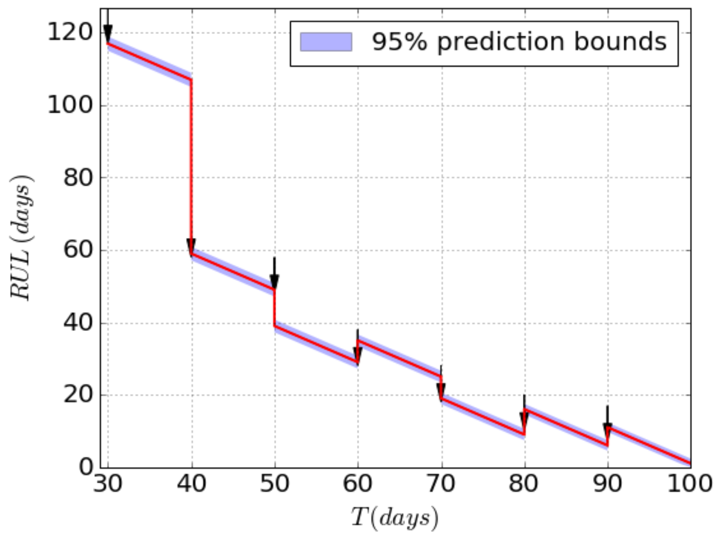


Figure 4.12 Remaining useful life prediction

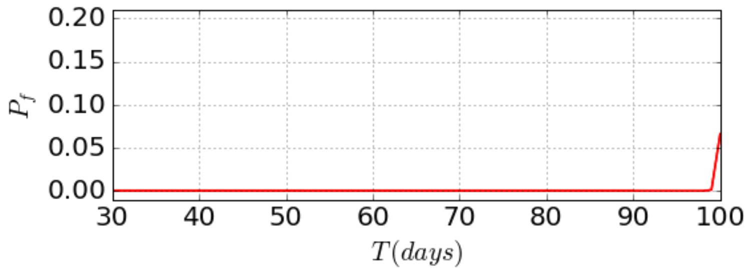


Figure 4.13 Probability of failure

4.6 Summary

This chapter developed a framework for applying big data analytics to uncertainty quantification in structural damage diagnosis and prognosis. The popular MapReduce approach was applied in the proposed framework for both the inverse and forward problems of UQ, and realized via Apache Spark. An ASR gel expansion model combined with FEA was used to perform prognosis, resulting in the prediction of ASR damaged area and remaining useful life along with probability bounds. Since this laboratory study did not generate very large amounts of data, MapReduce did not show the advantage in image processing. For practical concrete structures risk analysis, the big data issue will be more obvious and MapReduce will show greater benefits in scalability.

Future research needs to address several extensions. A major advantage of MapReduce will be in parallelizing FEA, since FEA is the most computationally expensive element in the aforementioned ASR prognosis. However, multiple commercial licenses are required to parallelize the FEA software via MapReduce; therefore methods to share license among slave nodes are worth exploring. Second, this chapter only considered diagnosis uncertainty and propagation of this uncertainty through prognosis (forward computation). In future research, other sources of uncertainties (both aleatory and epistemic) should be considered for comprehensive UQ analysis (e.g., epistemic uncertainty in the model parameters, uncertainty regarding the future loading, and the uncertainty in the prognosis model). Variance-based sensitivity analysis (Saltelli et al. 2008) is valuable in this regard; it can help to identify the dominant uncertainty sources affecting prognosis uncertainty and retain only those sources in the uncertainty quantification, thus significantly reducing the computational effort.

CHAPTER 5

BIG DATA ANALYTICS IN HIGH-DIMENSIONAL MODEL PARAMETERS CALIBRATION

5.1 Background

Chapter 4 developed a big data analytics approach for uncertainty quantification in structural diagnosis and prognosis, in which the structure's current state is diagnosed by data processing and Bayesian updating, and the structure's future state is predicted by the uncertainty propagation through the structural analysis model and damage growth model. The model which is used for prediction is important, and needs to be updated with the latest information; however, such updating is challenging when the observation data is large and the dimensionality of model parameters to be updated is high. The high dimensionality of model parameters often arises when their variability over space needs to be considered. Therefore the use of big data analytics in high-dimensional model parameters calibration is developed in this chapter.

Model calibration refers to the adjustment of model parameters so that the model output matches well with the field data. When full field observations are available (spatially or temporarily), different options are available for calibration. The most common approach is to consider the material properties are homogeneous, and to calibrate the parameters using observations at only a few locations. For example, Karabinis and Rousakis (2002) calibrated material parameters of carbon fiber-reinforced polymer (FRP) confined concrete by running only several experimental tests. Madsen (2003) estimated parameters of hydrological catchment model using observations from multiple locations. Lefèvre et al. (2003) calibrated thermal conductivity for a hot wire based on dc scanning thermal microscopy by measurements of different tip temperatures. Some researchers perform model calibration using dimension reduction methods. For example, Higdon et al. (2008a) used basis representations (e.g., principal components) to reduce the dimensionality of the problem and speed up the computations required for exploring the posterior distribution. Higdon et al. (2008b) also used singular value decomposition (SVD) to reduce the dimension. On the other hand, some researchers applied full field measurements to update the model parameters. For example, Roux and Bouchard (2015) calibrated a ductile damage model using measurements from the full displacement field. Nath et al. (2017) considered both methods mentioned above. First, random fields were utilized to account for the variability of model parameters over space and across the specimens, and SVD is applied for the purpose of dimension reduction. Then several observation spots were selected as optimum sensor locations by using the Kullback-Leibler (KL) divergence metric (Huang et al., 2007) to maximize the information gain. All the above approaches increase the computational efficiency, at the cost of accuracy.

Take finite element analysis as an example; traditionally we create a model with as small a number of parameters as possible, in order to save the computational effort. One example is that we consider the material’s property to be homogeneous in the whole model (of course different properties will be used when the model has parts of different materials). However, sometimes this cannot meet the researcher’s needs, when the object of interest consists of a heterogeneous material like concrete. Concrete is a composite material that is composed of coarse aggregate bonded together cement. Therefore, if we want to model structures built with concrete more accurately, material properties should be considered heterogeneous.

Only a few studies on the application of big data techniques to model calibration can be found in the literature. Humphrey et al. (2012) parallelized the calibration of parameters in watershed models, which was realized on a Windows Azure cloud computing platform. Zhang et al. (2014) realized cloud-based calibration of a hydrologic model on a Hadoop platform. These studies only parallelized the calibration process to particular applications (hydrological model), and did not handle large volumes of observations. In this chapter, a novel application of MapReduce to model calibration is presented. Here we focus on handling the big data issue in model calibration.

It is known that numerical models are sometimes too expensive to be repeatedly run during the calibration process, which calls for the construction and use of surrogate models. The training data collection and the training of the surrogate model are also parallelized in this chapter using MapReduce. The proposed methodology is general, and applies to variations over both space and time.

It can be observed that the main reason that researchers choose not to use full field observations to calibrate the spatially varying parameters of heterogeneous materials is due to computational cost. However, the price is loss of information and accuracy, since such a strategy implies that the model parameters do not vary over space and time. For the general, heterogeneous case where model parameters vary over space and time (e.g., material properties), full-field calibration would be high dimensional. Since calibration using full field observations is time consuming, parallel and distributed computing can help to reduce the time cost of data analytics, without causing any accuracy loss.

5.2 Bayesian Calibration of High-Dimensional Model Parameters

5.2.1 Overview of Bayesian Calibration

Consider a model G with inputs $\boldsymbol{a} = [a_1, a_2, \cdots, a_n]$, where n is the number of inputs, with known deterministic values or probability distributions and parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_p]$ that need to be calibrated, where p is the number of parameters. The model output $\boldsymbol{y}_m$, which is the prediction of the actual physical quantity $\boldsymbol{y}$, is given by

$$\boldsymbol{y}_m = G(\boldsymbol{a}, \boldsymbol{\theta}) \tag{5.1}$$

An observed output value from the experiment is denoted as $\boldsymbol{y}_{obs}$ with an observation error $\varepsilon_{obs} \sim N(0, \sigma_{obs}^2)$ where $N(\cdot,\cdot)$ stands for normal distribution. The experimental represented by observation $\boldsymbol{y}_{obs}$, model output $\boldsymbol{y}_m$ and true value of the true physical quantity $\boldsymbol{y}$ are related as

$$\boldsymbol{y}_{obs} = \boldsymbol{y} + \varepsilon_{obs} \tag{5.2}$$

$$\boldsymbol{y} = \boldsymbol{y}_m + \boldsymbol{\delta}(\boldsymbol{a}) \tag{5.3}$$

where $\boldsymbol{\delta}(\boldsymbol{a})$ is the model discrepancy term which is a function of the model inputs and needs to be calibrated. Different prior formulations of model discrepancy function were compared and evaluated by Ling et al. (2014). Combining Eq. (5.2) and Eq. (5.3), we have

$$\boldsymbol{y}_{obs} = \boldsymbol{y}_{\boldsymbol{m}} + \boldsymbol{\delta}(\boldsymbol{a}) + \varepsilon_{obs} \tag{5.4}$$

Using Bayes' theorem, the joint posterior distribution of the calibration parameters is obtained as

$$f(\boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta} | \boldsymbol{y}_{obs}) \propto f(\boldsymbol{y}_{obs} | \boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta}) f(\boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta}) \tag{5.5}$$

where $f(\boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta} | \boldsymbol{y}_{obs})$ is the joint probability density of $\boldsymbol{\theta}, \sigma_{obs}$ and $\boldsymbol{\delta}$, $f(\boldsymbol{y}_{obs} | \boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta})$ is the likelihood function, and $f(\boldsymbol{\theta}, \sigma_{obs}, \boldsymbol{\delta})$ is the prior probability density.

5.2.2 Calibration of High-dimensional Model Parameters

As mentioned in Sec. 5.1, high-dimensional model parameter calibration may be preferred for heterogeneous materials. In addition to variation in space, these parameters may also have variation across different specimens or realizations. For example, a slab might have a spatially varying parameter as shown in Fig. 5.1(a). Due to the inherent variability of the parameter, similar specimens may show different realization of the spatial variability in Fig. 5.1(b) and Fig. 5.1(c).

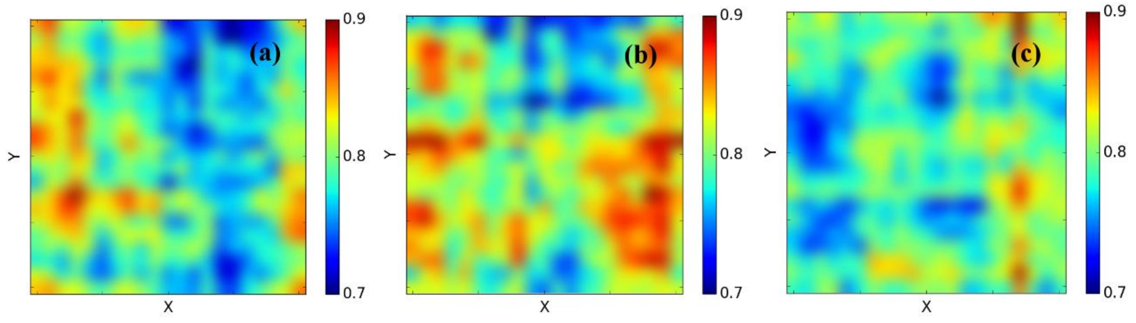


Figure 5.1 Probability of failure

Due to material variability, $\boldsymbol{\theta}$ may be defined as a function of locations $\boldsymbol{d}$. Since $\boldsymbol{d}$ is a large vector, the number of parameters $\boldsymbol{\theta}(\boldsymbol{d})$ is very large. Therefore, the calibration process is unaffordable using the conventional model calibration method, if it is impossible to develop a parametric representation of the model parameter over the spatial domain. Furthermore, full field observations (such as optical or thermal images) collected over a long time period will bring a in the big data issue. Thus in this chapter, we proposed big data techniques to solve the high-dimensional model parameter calibration in the presence of big data. The challenges in this problem can be summarized as follows:

- Due to the high-dimensional calibration parameter space, current Bayesian calibration techniques such as Markov Chain Monte Carlo (MCMC) simulation or particle filter (PF) require a large number of iterations or particles to converge. This will evaluate the likelihood function which is a function of the prediction model millions of times. Directly using the computer simulation model in the calibration process is computationally impossible. Even if the computational model is replaced with cheaper surrogate models, the required computational effort is still prohibitive for a single computer. The first challenge is therefore how to handle the computational effort issue in Bayesian calibration.
- Surrogate models are usually built to replace the original computer simulation model in Bayesian calibration. In order to compute the likelihood based on the surrogate modeling, surrogate models need to be functions of calibration parameters. When the dimension of calibration parameters is very high, current surrogate modeling methods will suffer from the curse of dimensionality. The second challenge is how to build surrogate models to replace the original computer simulation model in Bayesian calibration of high-dimensional model parameters.
- For problems with high-dimensional calibration parameters, the observations are also

high-dimensional. The third challenge is how to effectively utilize the high-dimensional observations in Bayesian calibration with the consideration of the correlations of observations over space and time.

5.3 Workflow of Model Calibration using MapReduce

In this section, we first provide a brief review of the MapReduce framework and Spark. Following that, we discuss how to address the challenges summarized in Sec.5.2 using the MapReduce framework and Spark.

In order to deal with all the three challenges discussed in Sec. 5.2.2, three levels of parallelization using MapReduce technique to Bayesian calibration. Fig. 4 shows the general procedure of the proposed model calibration framework. In order to save the computational cost, a surrogate model will be applied in the process of calibration. Thus first, the original simulation model is an FEA model, for which the design of experiment (DOE) will be performed. Then, using inputs and outputs for FEA, a surrogate model can be trained. Third, the observation data needs to be processed. Measurements from experiment or sensing data cannot be used directly, so preprocessing operations such as noise cancellation are always necessary. With the trained surrogate model and processed observations, the likelihood of the observation can be evaluated. Based on the likelihood function, a Bayesian calibration technique such as MCMC can be used to estimate the posterior distribution of calibration model parameters.

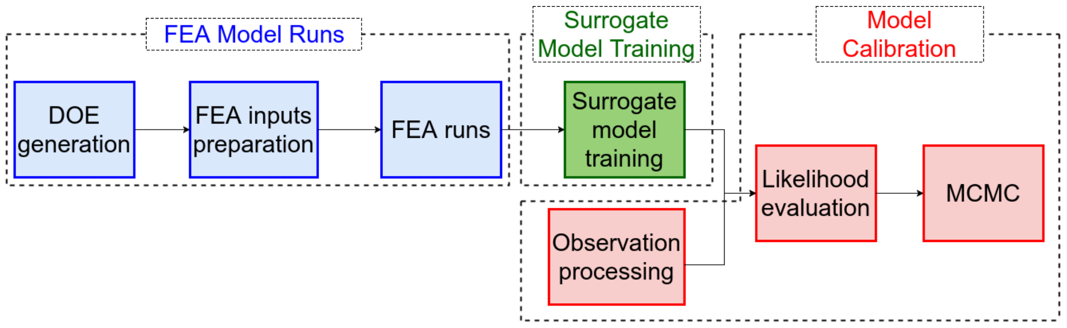


Figure 5.2 Workflow of model calibration

The three levels of parallelization can be summarized as: (1) Parallelization of FEA model runs (colored in blue): this level is used to generate training points for surrogate modeling; (2) Parallelization of surrogate model training (colored in green), and (3) high dimensional model calibration (colored in red). Since all three levels are potentially computationally expensive, application of MapReduce will be studied for each level. In the subsequent sections, we explain these three levels of parallelization in detail.

5.4 Level 1 Parallelization: MapReduce for FEA Model Runs

As mentioned previously, the surrogate model preparation has three steps: DOE generation, FEA model inputs preparation, and FEA model runs. Compare to all other steps, FEA model runs often consume most of the computational time. Thus a MapReduce parallelization methodology is developed for the evaluations of the FEA simulation model. Suppose n_s training points are needed, and therefore ns sets of parameter values will be generated, which are noted as $\theta_i, i = 1, \cdots, n_s$. Note here the number of variables depends on the number of parameters, and also depends on the spatial and temporal dimensions if the heterogeneity is considered.

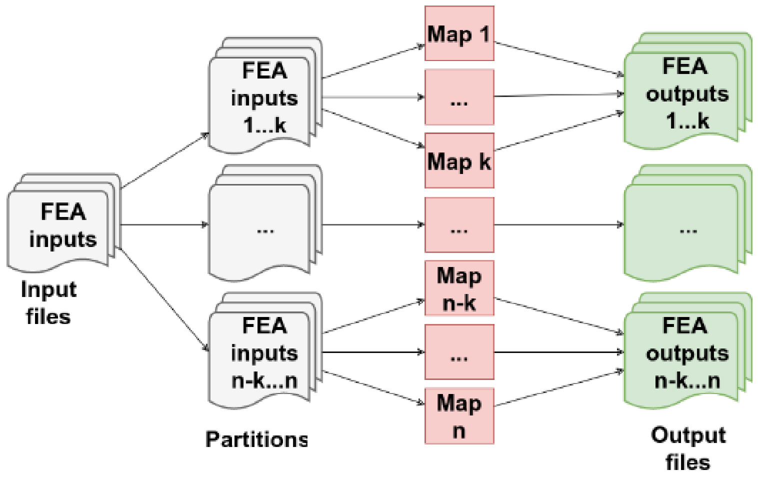
Fig. 5.3 (a) presents the pseudocode of the proposed parallelization procedure. The FEA input files are first divided into different partitions (each partition contains multiple files), and the FEA running command is then called inside the mapper function. For each FEA job, a separately processed result is obtained, without combination (Fig. 5.3). There are two steps in this pseudocode. First, a Map function is defined (‘mapper’), within which all the actual processing functions are defined. The argument ‘x’ is the data file id corresponding to the assigned tasks (FEA input files here) to be analyzed, which is assigned by the task manager. As discussed previously, since there is only the Map function, the input file can be mapped with any value (here we mapped ‘x’ to 0). In the second step, SparkContext, represents the connection to the cluster, which is the main class in Spark; ‘parallelize’ is the method to split the input files into *N* partitions; and ‘map’ is the method to call the Map function defined in the first step and to pass the input file to it. The ‘count’ method is used to count the number of outputs, which is used to trigger the parallelization.

```
Pseudocode 5.1:

function mapper(x);
   InputData = ReadData(x)
   OutputData = FEA(InputData)
   WriteData(OutputData)
   return (x, 0)

SparkContext(appName="myApp").parallelize(range(N),
N).map(mapper).count()
```

(a) MapReduce pseudocode



(b) Schematic description

Figure 5.3 Distributed computing of data processing

5.5 Level 2 Parallelization: Surrogate Model Training

5.5.1 Gaussian Process Surrogate Model with Spatially Varying Parameters

In order to build a surrogate model for the high-dimensional spatially varying response as a function of the calibration parameters, we first classify the calibration parameters into two categories: spatially constant calibration parameters ($\boldsymbol{\theta}^c$) and spatially varying calibration parameters ($\boldsymbol{\theta}^s(\boldsymbol{d})$). The spatially constant calibration parameters are used directly as the inputs of the surrogate model. Since the high-dimensional spatially varying parameters bring challenges to the surrogate model training due to the curse of dimensionality, $\boldsymbol{\theta}^s(\boldsymbol{d})$ is not directly used as input. Considering the fact that the response $\boldsymbol{y}$ at a spatial coordinate $\boldsymbol{d}$ is mainly affected by the responses and input parameters near this coordinate, we only use the $\boldsymbol{\theta}^s(\boldsymbol{d})$ in the neighboring locations of d as the input of $\boldsymbol{y}$ at $\boldsymbol{d}$.

Theoretically, for each spatial point, the parameters over the entire spatial domain should be used, since all the parameters will have contribution. However, it may not be necessary to consider parameters from all spatial points, depending on how fast the effects decrease with distance. Thus, we assume that for each spatial point, the response is only affected by its immediate neighbors. For instance, for the response at the location indicated with the blue star in Fig. 5.4, the parameters highlighted as red squares will be used as the inputs. Based on this assumption, the response at location $\boldsymbol{d}^{(i)}$ is approximated as

$$\boldsymbol{y}\left(\boldsymbol{d}^{(i)}\right) \approx \widehat{G}_i(\boldsymbol{\theta}^c, \boldsymbol{\theta}^s(\widehat{\boldsymbol{d}}^{(i)})) \tag{5.6}$$

where $\boldsymbol{d}^{(i)}$ is the i-th spatial coordinate, $\widehat{\boldsymbol{d}}^{(i)}$ stands for the neighboring locations of $\boldsymbol{d}^{(i)}$, and $\widehat{G}_i(\cdot)$ is the approximation model for the i-th location. In this chapter, we use the Gaussian process model reviewed in Sec. 2.6 to construct the approximation model $\widehat{G}_i(\cdot), \forall i = 1, 2, \cdots, m$, where m is the total number of spatial locations. Next, we discuss how to build these approximation models.

5.5.1.1 Generate Training Points

Defining $\boldsymbol{\beta} = \left[\boldsymbol{\theta}^c, \boldsymbol{\theta}^s\left(\boldsymbol{d}^{(1)}\right), \boldsymbol{\theta}^s\left(\boldsymbol{d}^{(2)}\right), \cdots, , \boldsymbol{\theta}^s\left(\boldsymbol{d}^{(m)}\right)\right]$, we first generate n_s training points for $\boldsymbol{\beta}$. For each training point $\boldsymbol{\beta}^{(i)}$, the response field is obtained using the original simulation model G as below:

$$\left[\boldsymbol{y}\left(\boldsymbol{d}^{(1)}, \boldsymbol{\beta}^{(i)}\right), \boldsymbol{y}\left(\boldsymbol{d}^{(2)}, \boldsymbol{\beta}^{(i)}\right), \cdots, , \boldsymbol{y}\left(\boldsymbol{d}^{(m)}, \boldsymbol{\beta}^{(i)}\right)\right] = G(\boldsymbol{\beta}^{(i)}) \tag{5.7}$$

where $\boldsymbol{y}(\boldsymbol{d}^{(j)}, \boldsymbol{\beta}^{(i)})$ denotes the response at the j-th spatial location of the i-th training point $\boldsymbol{\beta}^{(i)}$. It should be noted that the output is a field response (as indicated in Eq. (5.7)) for given training point $\boldsymbol{\beta}^{(i)}$.

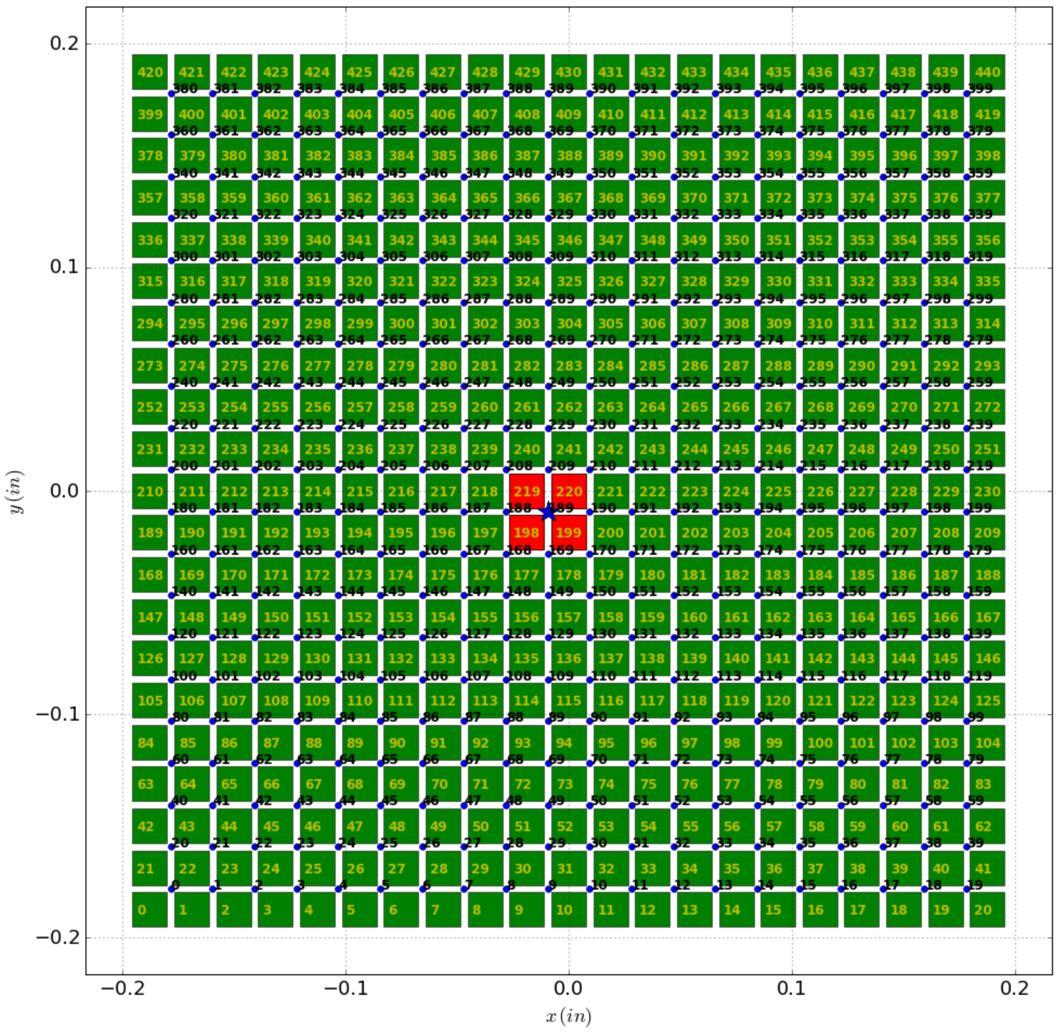


Figure 5.4 21 × 21 calibration grid and 20 × 20 observation points

After performing simulations at all the training points, a data matrix is obtained as $\boldsymbol{y}_{total} = \{\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}), i = 1, \cdots, m; j = 1, \cdots, n_s\}$. Note that the above matrix is obtained by reorganizing the spatial response into a one-dimensional array (i.e., $\boldsymbol{y}(\boldsymbol{d}, \boldsymbol{\beta}^{(j)}) = [\boldsymbol{y}(\boldsymbol{d}^{(1)}, \boldsymbol{\beta}^{(j)}), \boldsymbol{y}(\boldsymbol{d}^{(2)}, \boldsymbol{\beta}^{(j)}), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(m)}, \boldsymbol{\beta}^{(j)})]$ denotes the responses at all the spatial locations).

5.5.1.2 Surrogate Modeling

With the training data matrix $\boldsymbol{y}_{total}$, we then build surrogate models for response at different locations based on the assumption made in Eq. (5.6). For the i-th spatial location, we extract the training input values as $\widehat{\boldsymbol{\beta}}_{in}^{(i)} = [\widehat{\boldsymbol{\beta}}_1^{(i)}, \widehat{\boldsymbol{\beta}}_2^{(i)}, \cdots, \widehat{\boldsymbol{\beta}}_s^{(i)}]$, where $\widehat{\boldsymbol{\beta}}_j^{(i)} = [\boldsymbol{\theta}^c, \boldsymbol{\theta}^s(\widehat{\boldsymbol{d}}^{(i)})]$ is the j-th training point for the i-th location. The corresponding training output values are $\boldsymbol{y}_{out}^{(i)} = [\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(1)}), \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(2)}), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(s)})]$. Based on the training points $[\widehat{\boldsymbol{\beta}}_m^{(i)}, \boldsymbol{y}_{out}^{(i)}]$, the approximated model $\widehat{G}_i(\cdot)$ can be built using the Gaussian process surrogate modeling technique. However, when the simulation model is executed over time, $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ is a time-dependent trajectory even for a specific spatial location $\boldsymbol{d}^{(i)}$ and we have $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) = [\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_1), \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_2), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_{n_t})]$, where n_t is the number of time instants. This introduces extra challenge to the surrogate modeling. Next, we investigate how to address this issue using singular value decomposition (SVD).

5.5.3 Gaussian Process Surrogate Model with Temporal Correlation

Singular value decomposition (SVD) is a multivariate statistical method to describe a large amount of high-dimensional data by mapping to a low-dimensional space (Chatterjee 2000). SVD can be used for handling the temporal correlation of the response. Given m data points over the spatial domain $\boldsymbol{\Omega}$ for n_t time domain realizations, a data matrix can be collected as follows:

$$\boldsymbol{\omega} = [\boldsymbol{\omega}(\xi_1), \boldsymbol{\omega}(\xi_2), \cdots, \boldsymbol{\omega}(\xi_s)]^T = \begin{bmatrix} \omega(t_1, \xi_1) & \omega(t_1, \xi_2) & \cdots & \omega(t_1, \xi_{n_t}) \\ \omega(t_2, \xi_1) & \omega(t_2, \xi_2) & \cdots & \omega(t_2, \xi_{n_t}) \\ \vdots & \vdots & \ddots & \vdots \\ \omega(t_{n_t}, \xi_1) & \omega(t_{n_t}, \xi_2) & \cdots & \omega(t_{n_t}, \xi_{n_t}) \end{bmatrix}^T \quad (5.7)$$

where $\boldsymbol{\omega}(\xi_i) = [\omega(t_1, \xi_i), \omega(t_2, \xi_i), \cdots, \omega(t_{n_t}, \xi_i)]$ is the i-th realization..

This large amount of high-dimensional data can be mapped to a low-dimensional space by using SVD as $\boldsymbol{\omega} = \boldsymbol{V}\boldsymbol{M}\boldsymbol{U}^T$, where $\boldsymbol{V}$ is a $s \times n_t$ matrix, $\boldsymbol{U}$ is a $n_t \times n_t$ orthogonal matrix and $\boldsymbol{M}$ is a $n_t \times n_t$ rectangular diagonal matrix with non-negative real numbers $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_m]$ on the diagonal. Here we donate $\boldsymbol{\gamma} = \boldsymbol{V}\boldsymbol{M}$, the matrix can be constructed as

$$\boldsymbol{\omega}(\cdot, \xi_i)^T \approx \sum_{j=1}^{r} \gamma_{ij} \boldsymbol{U}_j \quad (5.8)$$

where $\boldsymbol{\omega}(\cdot, \xi_i)^T$ is the i-th row of $\boldsymbol{\omega}$, γ_{ij} is the element of $\boldsymbol{\gamma}$ at i-th row and j-th column, $\boldsymbol{U}_j$ is the j-th important feature vector used to approximate $\boldsymbol{\omega}$, and r is the number of important features used. The number of features r is determined based on the magnitudes of the singular values $\boldsymbol{\lambda}$ (Xu, 1998).

Based on SVD, the response at spatial location $\boldsymbol{y}_{all}^{(i)} = \{\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k), j = 1, 2, \cdots, s, k = 1, 2, \cdots, n_t\}$ is reconstructed as

$$\boldsymbol{y}\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k\right) \approx \boldsymbol{\mu}_i(t_k) + \sum_{q=1}^{r} \gamma_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right) U_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{t}_k\right), \forall j = 1, 2, \cdots, s; k = 1, 2, \cdots, n_t \quad (5.9)$$

where $\boldsymbol{\mu}_i(t_k)$ is the mean value at location $\boldsymbol{d}^{(i)}$ at time instant t_k, $\gamma_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right)$ is the q-th latent response of spatial location $\boldsymbol{d}^{(i)}$ for the j-th training point, and $\boldsymbol{U}_q\left(\boldsymbol{d}^{(i)}, t_k\right)$ is the value of the q-th important feature $\boldsymbol{U}_q$ of $\boldsymbol{d}^{(i)}$ at time instant t_k.

Eq. (5.9) shows that the variation in the high-dimensional response mainly comes from the variation in $\boldsymbol{\gamma}\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right) = [\gamma_1\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)}\right), \gamma_2\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)}\right), \cdots, \gamma_r\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)}\right)]$, which denotes the value of $\boldsymbol{\gamma}$ of the response at $\boldsymbol{d}^{(i)}$ for the j-th training point. The dimension of

$\boldsymbol{\gamma}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ is usually much smaller than that of the response $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) = [\boldsymbol{y}(\boldsymbol{d}^{(i)}, t_1), \boldsymbol{y}(\boldsymbol{d}^{(i)}, t_2), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(i)}, t_{n_t})]$.

With the training points $\boldsymbol{\gamma}_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right), \forall q = 1, 2, \cdots, r;\ j = 1, 2, \cdots, s$ and $\widehat{\boldsymbol{\beta}}_{in}^{(i)} = [\widehat{\boldsymbol{\beta}}_1^{(i)}, \widehat{\boldsymbol{\beta}}_2^{(i)}, \cdots, \widehat{\boldsymbol{\beta}}_s^{(i)}]$, we construct surrogate model for $\boldsymbol{\gamma}_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right), \forall q = 1, 2, \cdots, r$. After substituting $\boldsymbol{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ with surrogate model $\boldsymbol{\gamma}_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right)$, Eq. (5.9) becomes:

$$\boldsymbol{y}\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k\right) \approx \boldsymbol{\mu}_{\boldsymbol{i}}(t_k) + \sum_{q=1}^{r} \hat{\gamma}_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right) U_q\left(\boldsymbol{d}^{(i)}, t_k\right), \forall j = 1, 2, \cdots, s; k = 1, 2, \cdots, n_t \qquad (5.10)$$

where $\widehat{\boldsymbol{\gamma}}_q\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}\right)$ stands for the q-th surrogate model associated with the spatial location $\boldsymbol{d}^{(i)}$. Note that $\gamma_1, \gamma_2, \cdots, \gamma_r$ are not the original responses but latent responses obtained through SVD.

5.5.3 MapReduce for Surrogate Model Training

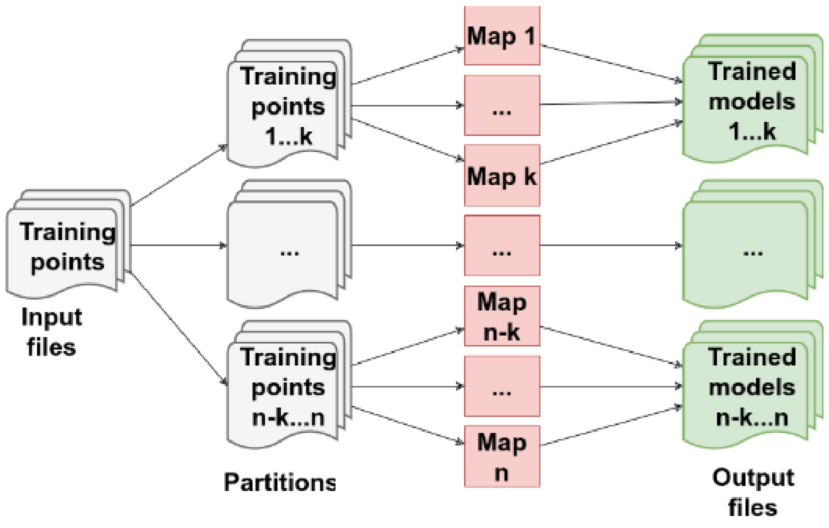
The MapReduce implementation of surrogate model training is shown in the pseudocode in Fig. 5.5. Here ‘x’ is the id of the file which is assigned to a particular slave node that the code is running on. Suppose there are n pairs of inputs and outputs, and surrogate models will be obtained after the parallel runs on slave nodes. The actual surrogate model training function will be called inside the mapper function. Each mapper will read one set of inputs and outputs, and save the trained model on to the disk. Note here that for each surrogate model, the inputs and outputs could be a vector, depending on the problem.

```
Pseudocode 5.2:

function mapper(x):
    InputData = ReadData(x)
    SurrogateModes = SurrogateTrain(InputData)
    WriteData(SurrogateModel)
    return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()
```

(a) MapReduce pseudocode



(b) Schematic description

Figure 5.5 Distributed computing of surrogate model training

5.6 Level 3 Parallelization: MapReduce for High-dimensional Model Calibration

5.6.1 Bayesian Calibration of Spatially Varying Parameters

We will now discuss how to perform Bayesian calibration for the spatially heterogeneous model parameters based on the above developed surrogate model. As mentioned in Sec. 5.5.1, we define the calibration parameters $\boldsymbol{\beta} = [\boldsymbol{\theta}^{\boldsymbol{c}}, \boldsymbol{\theta}^{s}(\boldsymbol{d}^{(1)}), \boldsymbol{\theta}^{s}(\boldsymbol{d}^{(2)}), \cdots, \boldsymbol{\theta}^{s}(\boldsymbol{d}^{(m)})]$. We also define $\boldsymbol{y}_{obs}^{all} = [\boldsymbol{y}_{obs}(\boldsymbol{d}^{(1)}), \boldsymbol{y}_{obs}(\boldsymbol{d}^{(2)}), \cdots, \boldsymbol{y}_{obs}(\boldsymbol{d}^{(m)})]$ where $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}) = [\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}, t_1), \boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}, t_2), \cdots, \boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}, t_{n_t})]^T$ is the observation at the i-th spatial location. A critical step is obtaining the posterior distributions $f(\boldsymbol{\beta}|\boldsymbol{y}_{obs}^{all})$ is the evaluation of the likelihood function $\boldsymbol{L}(\boldsymbol{y}_{obs}^{all}|\boldsymbol{\beta})$, which is computed based on the assumption made in Eq. (5.6) as follows

$$\boldsymbol{L}(\boldsymbol{y}_{obs}^{all}|\boldsymbol{\beta}) = \prod_{i=1}^{m} \boldsymbol{L}(\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)})|\boldsymbol{\beta}) \tag{5.11}$$

in which $\boldsymbol{L}(\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)})|\boldsymbol{\beta})$ is the probability of observing $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)})$ for given $\boldsymbol{\beta}$.

For given $\boldsymbol{\beta}$ and time instants $\boldsymbol{t}$, the observation $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}, t)$ at spatial location $\boldsymbol{d}^{(i)}$ can be expressed as

$$\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}, \boldsymbol{t}) = \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t}) + \boldsymbol{\delta}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t}) + \varepsilon_{obs}(\boldsymbol{d}^{(i)}, \boldsymbol{t}) \tag{5.12}$$

in which $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t})$ is the model prediction at spatial location $\boldsymbol{d}^{(i)}$ and time instant $\boldsymbol{t}$, $\boldsymbol{\delta}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t})$ is the model discrepancy term due to numerical approximation and underlying missing physics, and $\varepsilon_{obs}(\boldsymbol{d}^{(i)}, \boldsymbol{t})$ is observation error which is usually assumed to be a Gaussian random variable.

Since the prediction model $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, t)$ is approximated by surrogate models in Sec.

5.5.1.1, we can rewrite Eq. (5.12) as

$$\boldsymbol{y}_{obs}\left(\boldsymbol{d}^{(i)},\boldsymbol{t}\right) \approx \hat{y}\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) + \boldsymbol{\delta}\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) + \varepsilon_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t}) \qquad (5.13)$$

where $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t})$ is the approximate model (i.e. surrogate model) of $\boldsymbol{y}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t)$ is given by

$$\hat{\boldsymbol{y}}\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) \approx \mu_i(\boldsymbol{t}) + \sum_{q=1}^{r}\hat{\gamma}_q\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right)U_q(\boldsymbol{d}^{(i)},\boldsymbol{t}) \qquad (5.14)$$

Eq. (5.14) implies that $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t})$ is a linear combination of surrogate model $\hat{\gamma}_q\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right), q = 1, 2, \cdots, r$. Since the prediction of $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta})$ for given $\hat{\boldsymbol{\beta}}$ follows a normal distribution, the prediction of $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t)$ also follows a normal distribution. Note $\hat{\boldsymbol{\beta}} = [\boldsymbol{\theta}^c, \boldsymbol{\theta}^s(\hat{\boldsymbol{d}})]$ is a subset of $\boldsymbol{\beta}$. The mean and variance of $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t})$ are given by

$$\boldsymbol{\mu}_y\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) \approx \boldsymbol{\mu}_i(\boldsymbol{t}) + \sum_{q=1}^{r}\mu_{\hat{\gamma}_q}\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right)U_q(\boldsymbol{d}^{(i)},\boldsymbol{t}) \qquad (5.15)$$

$$\boldsymbol{\sigma}_y^2(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}) \approx \sum_{q=1}^{r}\sigma_{\hat{\gamma}_q}^2\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right)U_q^2(\boldsymbol{d}^{(i)},t) \qquad (5.16)$$

When the discrepancy term $\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t)$ is modeled as a Gaussian process model, the discrepancy term also follows normal distribution for given $\boldsymbol{\beta}$ and $\boldsymbol{t}$. Since $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t})$ is a linear function of $\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t})$, $\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t})$ and $\varepsilon_{obs}(\boldsymbol{d}^{(i)},t)$, $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t})$ also follows a normal distribution. Then mean and variance of $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t})$ is given by

$$\boldsymbol{\mu}_{\boldsymbol{y}_{obs}}\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) \approx \boldsymbol{\mu}_i(\boldsymbol{t}) + \sum_{q=1}^{r}\mu_{\hat{\gamma}_q}\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right)U_q\left(\boldsymbol{d}^{(i)},\boldsymbol{t}\right) + \boldsymbol{\mu}_{\boldsymbol{\delta}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}) \qquad (5.17)$$

$$\boldsymbol{\sigma}_{\boldsymbol{y}_{obs}}^2\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) \approx \sum_{q=1}^{r}\sigma_{\hat{\gamma}_q}^2\left(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}}\right)U_q^2\left(\boldsymbol{d}^{(i)},t\right) + \sigma_{\delta}^2\left(\boldsymbol{d}^{(i)},\boldsymbol{\beta},\boldsymbol{t}\right) + \sigma_{obs}^2(\boldsymbol{d}^{(i)},\boldsymbol{t}) \qquad (5.18)$$

The above equations imply that the uncertainty in the surrogate models $\hat{\gamma}_q(\boldsymbol{d}^{(i)},\boldsymbol{\beta})$, $q = 1, 2, \cdots, r$ will propagate to the uncertainty of $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t})$. In addition, the covariance between $\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},\boldsymbol{t})$ at time instants t_j and t_k is computed by

$$\boldsymbol{\Sigma}_i(j,k) = E\big([\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j) + \boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j) + \varepsilon_{obs}(\boldsymbol{d}^{(i)},t_j)][\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k) + \boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k) + \varepsilon_{obs}(\boldsymbol{d}^{(i)},t_k)]\big) - E([\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j) + \boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j) + \varepsilon_{obs}(\boldsymbol{d}^{(i)},t_j)])E([\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k) + \boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k) + \varepsilon_{obs}(\boldsymbol{d}^{(i)},t_k)]) \quad (5.19)$$

in which $E(\cdot)$ stands for "expectation".

After simplification, we have

$$\boldsymbol{\Sigma}_i(j,k) = E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right) - E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\right)E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right) + E(\delta(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\delta(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k) - E\left(\delta(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\right)E(\delta(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)) \quad (5.20)$$

In the above equation, $E\left(\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right) - E\left(\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\right)E\left(\boldsymbol{\delta}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right)$ is the covariance of the model discrepancy at different time instants. If the model discrepancy terms are assumed to be independent of time, we have

$$\boldsymbol{\Sigma}_i(j,k) = E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right) - E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_j)\right)E\left(\hat{\boldsymbol{y}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_k)\right) \quad (5.21)$$

Substituting Eq. (5.14) into Eq. (5.21) yields

$$\boldsymbol{\Sigma}_i(j,k) = \sum_{q=1}^{r}\sigma^2_{\hat{\gamma}_q}(\boldsymbol{d}^{(i)},\hat{\boldsymbol{\beta}})U_q(\boldsymbol{d}^{(i)},t_j)U_q(\boldsymbol{d}^{(i)},t_k), \forall j,k = 1,2,\cdots,n_t \quad (5.22)$$

Based on Eqs. (5.17), (5.18) and (5.22), $L(\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)})|\boldsymbol{\beta})$ is then computed by

$$L(\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)})|\boldsymbol{\beta}) = L([\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},t_1),\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},t_2),\cdots,\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)},t_{n_t})]|\boldsymbol{\beta}) = \frac{1}{(2\pi)^{\frac{n_t}{2}}\sqrt{|\boldsymbol{\Sigma}_i|}}\exp(-\frac{1}{2}(\boldsymbol{y}_{obs}(\boldsymbol{d}^{(i)}) - \boldsymbol{\mu}_i)^T\boldsymbol{\sigma}_i^{-1}(\boldsymbol{d}^{(i)} - \boldsymbol{\mu}_i)) \quad (5.23)$$

where $\boldsymbol{\mu}_i = [\boldsymbol{\mu}_{\boldsymbol{y}_{obs}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_1),\boldsymbol{\mu}_{\boldsymbol{y}_{obs}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_2),\cdots,\boldsymbol{\mu}_{\boldsymbol{y}_{obs}}(\boldsymbol{d}^{(i)},\boldsymbol{\beta},t_{n_t})]$ and $\boldsymbol{\Sigma}_{\text{i}}$ is the covariance matrix with $\boldsymbol{\Sigma}_{\text{i}}(j,k)$ is given by Eq. (5.22) and diagonal elements given by Eq. (5.18).

With Eqs. (5.23) and (5.17), $L(\boldsymbol{y}_{obs}^{all}|\boldsymbol{\beta})$ can be computed for given $\boldsymbol{\beta}$. The posterior distribution $f(\boldsymbol{\beta}|\boldsymbol{y}_{obs}^{all})$ can then be estimated using Bayesian inference as

$$f(\boldsymbol{\beta}|\boldsymbol{y}_{obs}^{all}) \propto L(\boldsymbol{y}_{obs}^{all}|\boldsymbol{\beta})f(\boldsymbol{\beta}) \tag{5.24}$$

5.6.2 MapReduce for Data Processing

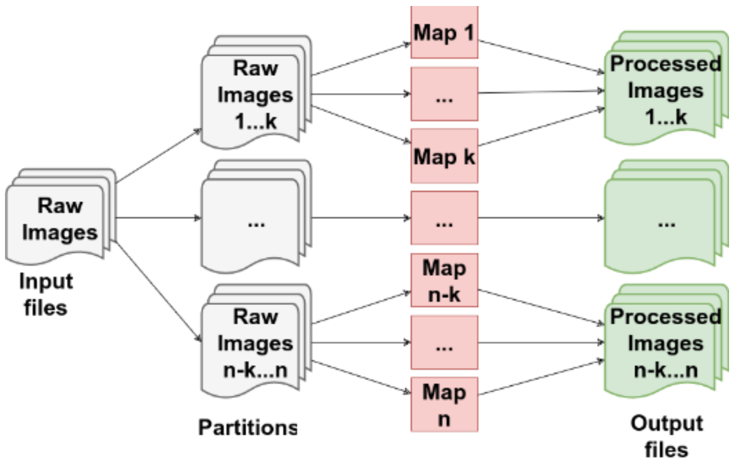
The field data being observed can be of different formats, e.g. images, time histories, and other recorded measurements. Usually those raw data cannot be used for calibration directly because of noice, thus data processing is needed before feeding into the model. Here we consider image processing as an example for the purpose of illustration. Note that the general parallelization procedure of image can be applied similarly to other data formats. In the case of thermal image processing, the common procedure is: cropping, baseline removal, and noise cancellation. The mapper function and the schematic description are shown in Fig. 5.6. In the mapper function, ‘x’ is the id of the file assigned to the slave node on which this function is running on. Note that in the mapper, all the processing steps will be executed sequentially. ‘ReadData’ function in Pseudocode 5.3 is used for reading of inputs, and the data is stored in variable ‘InputData’. ‘Cropping’ is a function for cropping the images, which stores the cropped image pixel values into the variable ‘CroppedImage’. Finally the ‘NoiseCancel’ function will be called to cancel the noise in the image and save the output onto disk.

```
Pseudocode 5.3:

function mapper(x):
    InputData = ReadData(x)
    CroppedImage = Cropping(InputData)
    NoiseCancelledImage = NoiseCancel(CroppedImage)
    WriteData(NoiseCancelledImage)
    return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()
```

(a) MapReduce pseudocode



(b) Schematic description

Figure 5.6 Distributed computing of data processing

5.6.3 MapReduce for Likelihood Evaluation

The likelihood evaluation step is the most expensive step in Bayesian calibration. The parallelization of the likelihood evaluation is realized inside the MCMC MapReduce algorithm (Fig. 5.7).

5.6.4 MapReduce for MCMC

The basic idea of MCMC parallelization is to divide the observations into M splits, with each node taking one partition to provide samples of the posterior distribution. The prior distribution of the variable of interest will be updated using the equation:

$$p_m(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})^{\frac{1}{M}} p(\boldsymbol{x}^{n_m}|\boldsymbol{\theta}) \tag{5.17}$$

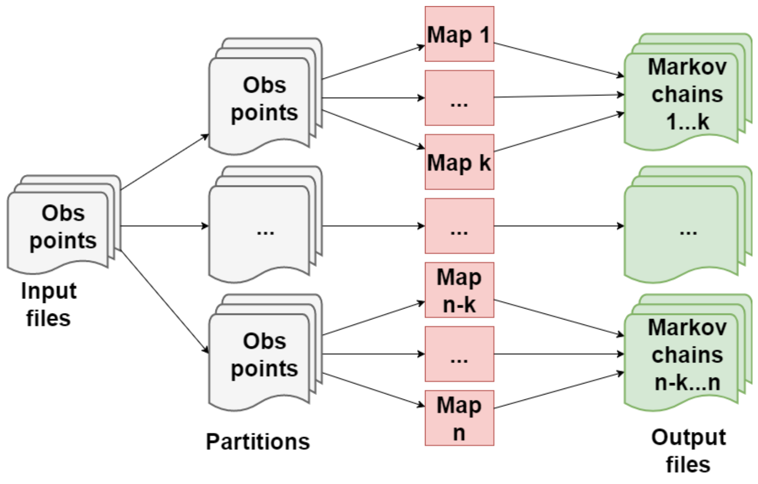
After all nodes complete their tasks, all the sub-posterior samples from each nodes will be combined to produce samples for an estimate of the sub-posterior density product $p_1 p_2 \cdots p_M(\boldsymbol{\theta})$, which is proportional to the full data posterior, i.e., $p_1 p_2 \cdots p_M(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\boldsymbol{x}^N)$.

```
Pseudocode 5.4:

function mapper(x):
    InputData = ReadData(x)
    OutputData = MCMC_Sampling(InputData)
    SaveSamples(OutputData)
    return (x, 0)

SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()
```

(a) MapReduce pseudocode



(b) Schematic description

Figure 5.7 Distributed computing of parameter calibration by MCMC

A Map function is defined ('mapper'), within which all the actual functions are defined (ReadData(), MCMC_Sampling(), and SaveSamples()). As shown in Fig. 5.5, the sampling process is executed on the slave nodes, while posterior integration is done after all particles and weights are saved from the slave nodes. SparkContext and count() function are used the same way as in Fig. 5.3. ReadData() is the function used to read observation data and parameters, and followed by MCMC_Sampling(), which is the function to perform the sampling. SaveSamples() is the function used to save all subset of MCMC chains.

In summary, the steps for calibration of high-dimensional model parameters using big data analytics are: (1) parallelize FEA model runs; (2) parallelize the training of surrogate models; (3) parallelize model calibration.

5.7 Numerical Example

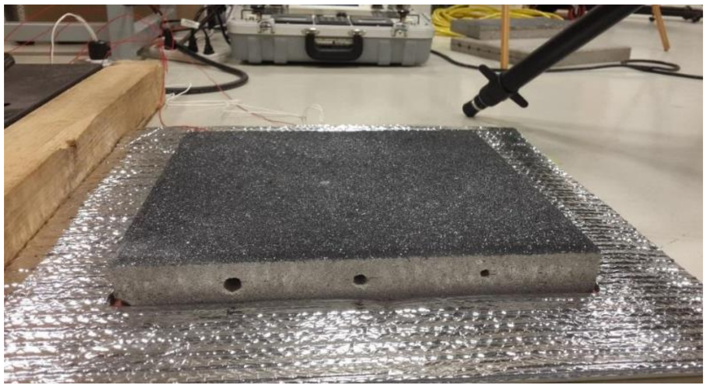
The proposed methodology for big data analytics in model calibration with heterogeneous materials is illustrated for the calibration of thermal conductivity in a concrete structure. A concrete structure with damage is considered, where the damage is simulated by drilled holes (Fig. 5.8 (a)) thus introducing heterogeneity. We need to use different conductivity coefficient values at different locations in order to use in future prognosis of the structure. In a realistic structure, the damaged area could be quite irregular; thus an averaged value or a parametric random field representation of property variation may not be feasible. As a result, we may need to discretize the entire domain into many sub-domains (consistent with the FEA model) and calibrate the property for each sub-domain. In that case, calibration becomes a high-dimensional problem if many sub-domains need to be considered.

5.7.1 Collection of Observation Data for Calibration

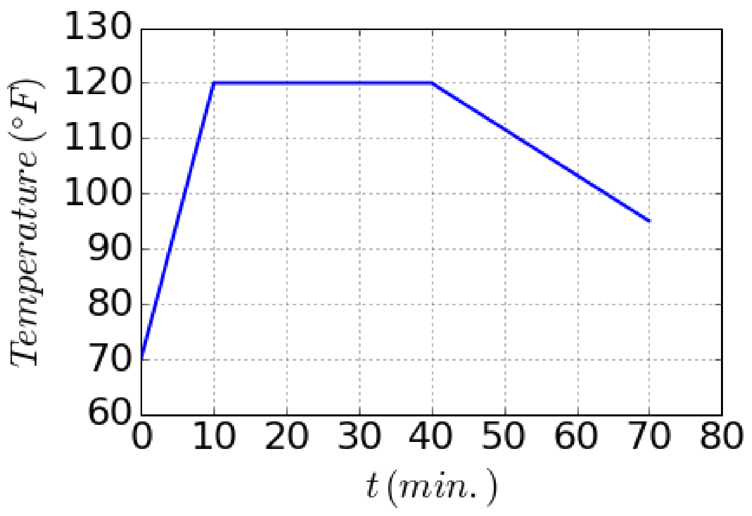
5.7.1.1 Experimental Setup

The concrete slab is placed on a thermal blanket which is heated according to a predefined profile (Fig. 5.8). The top surface temperature is obtained after processing thermography images captured by an infrared camera. Note that since the material is highly heterogeneous, we are calibrating the thermal conductivity in different locations on the top surface. Since we can only observe the thermography image on the top surface, and also the thickness is small compared to its length and width, it is reasonable to assume the thermal conductivity does not vary along the thickness.

To mimic damage and introduce heterogeneity, holes of $1/2$ inch, $3/8$ inch, and $5/16$ inch diameter (all of them 4.45 inch deep) were drilled into the side of the concrete slab, as shown in Fig. 5.8 (a). The thermal loading history is shown in Fig. 5.8 (b), with heating, stable, and cooling periods. In realistic situations, concrete damage could be of many types (physical, chemical, and mechanical), due to various causes such as freeze-thaw, chloride penetration, alkali-silica reaction etc. Temperature, humidity, and the properties of the concrete constituents (cement, aggregates, reinforcing steel, water content, and chemical admixtures) play a crucial role in the evolution of various types of damage. Under such damage (of unknown geometry), it is only appropriate to model the material as heterogeneous.



(a) The specimen to be monitored with thermal blanket below



(b) The thermal loading history being applied (scaled values)

Figure 5.8 Experiment setting

5.7.2 Finite Element Model

Fig. 5.9 shows the meshed FEA model implemented in commercial software Abaqus, with 3009 nodes and 7038 thermal-coupled elements (994 linear hexahedral elements and 6044 linear tetrahedral elements). The thermal conductivity coefficients at different spatial locations on the top surface need to be calibrated. In the FEA model, the spatial locations are represented as a 21×21 grid as shown in Fig. 5.4. For each calibration block location, the thermal conductivity is considered to be constant. We use $400\ (= 20 \times 20)$ observation points on the top surface, and assume that the temperature value at each observation point is affected by only the four neighboring blocks. For example, observation point 189 is affected by blocks 168, 169, 188 and 189 (Fig. 5.4).

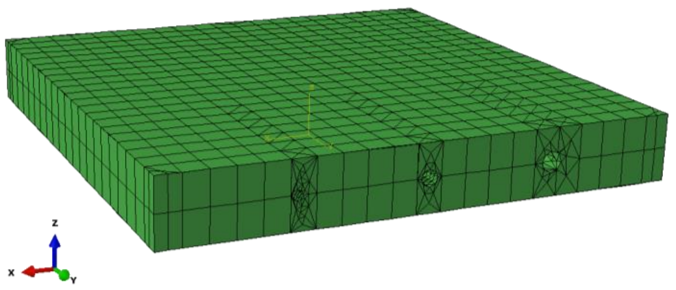


Figure 5.9 FEA model for concrete slab

Table 5.1 Concrete model parameters

Parameter	**Unit**	**Value**
Elasticity	Pa	40×10^9
Poisson’s ratio	$-$	0.15
Thermal expansion	$1/K$	7.4×10^{-6}
Specific heat	$J/Kg \cdot K$	880

Table 5.1 shows the concrete model parameters except thermal conductivity k. Thermal conductivity is considered to be in the range of $[0.8, 2.5]$ $W \cdot m^{-1} \cdot K^{-1}$. Since the FEA model is too expensive for Bayesian calibration, we use a surrogate model to replace it. Training points of the surrogate model are obtained using a Latin-hypercube design, with 5 conductivity values in each block. Thus for each observation point, the number of DOE points are $625 = 5 \times 5 \times 5 \times 5$, since k values at four neighboring blocks are used as inputs to the surrogate model for each spatial location. One example realization of training inputs is shown in Fig. 5.10 (the axis values are block indices in x and y direction). Since in

each FEA run, the temperature at all locations can be obtained at the same time, the total number of FEA runs will be 625. These 625 runs can be parallelized via MapReduce as described in Sec. 5.3.

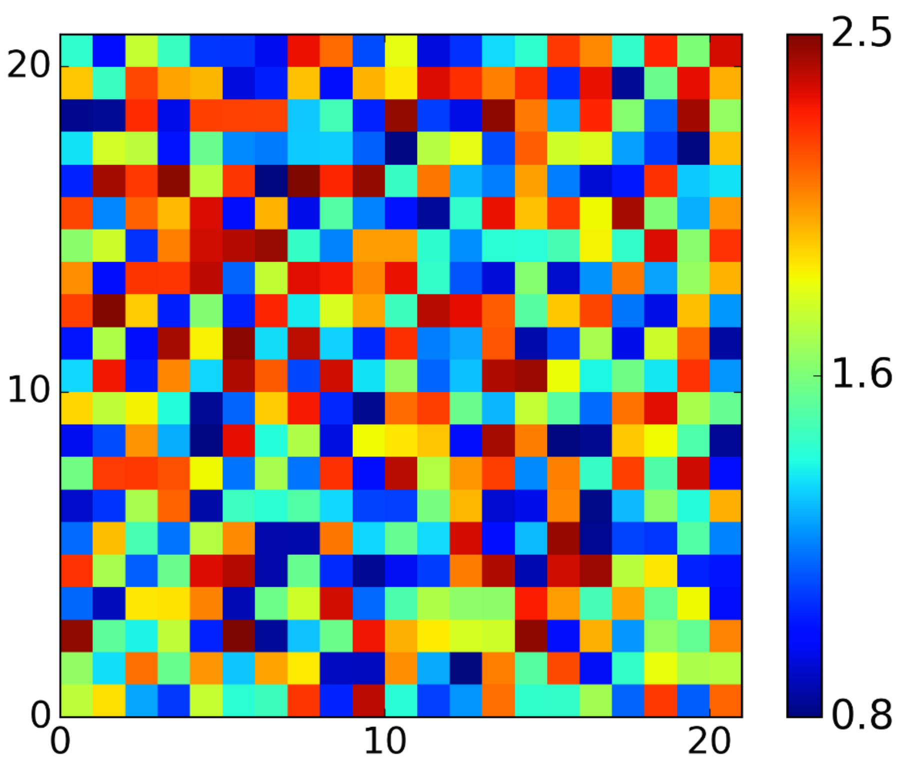


Figure 5.10 Example realization of k values for one training point

5.7.3 Surrogate Model Training

Based on the inputs (conductivity values) and outputs (nodal temperature values in each run), the Gaussian process surrogate model can be obtained. One example output is shown in Fig. 5.11. For each FEA output, we will have a series of output for 70 time steps (70 mins). For each spatial location i, if we create a surrogate model for each time step, we will lose the correlation between each time step. In order to capture the correlation over time, and also to reduce the dimension, singular value decomposition (SVD) is applied. Following Eq. (5.8), where $\boldsymbol{\omega}$ is the temperature output at each location for all 625 training points (625×70), $\boldsymbol{V}$ is the left singular vectors (70×70), $\boldsymbol{M}$ is the matrix of singular values (70×70), and $\boldsymbol{U}$ is the matrix of right singular vectors (70×70). Here we choose only the first two components, which means we will use the first two columns of $\boldsymbol{VM}$, and the first two rows of $\boldsymbol{U}$. Thus we have two bases $\boldsymbol{U}_0$ and $\boldsymbol{U}_1$ are used here as an example (Fig. 5.12(a)), and the corresponding coefficient for each DOE output will have a dimension of 1×2 (Fig. 5.12 (b)). Fig. 5.12 (c) shows that the 2-components SVD captures the temporal history very well.

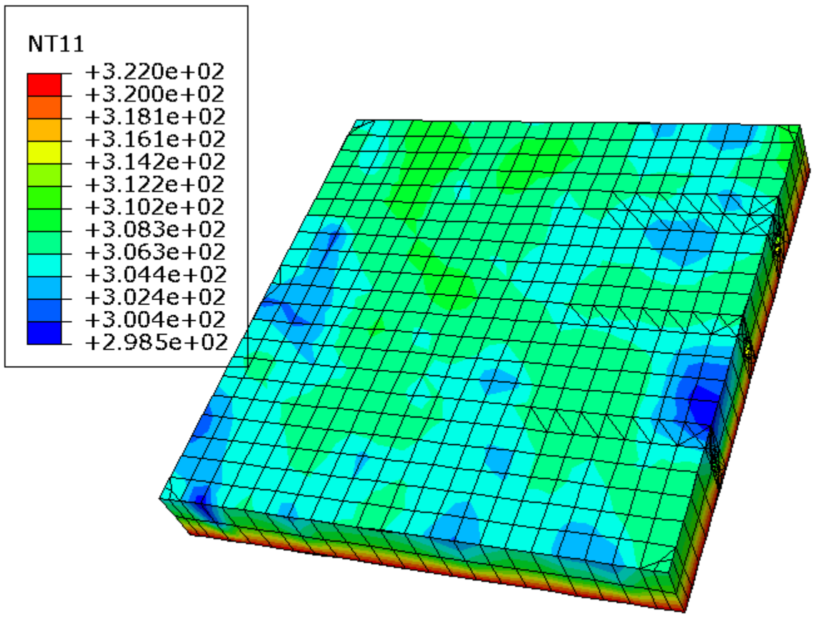
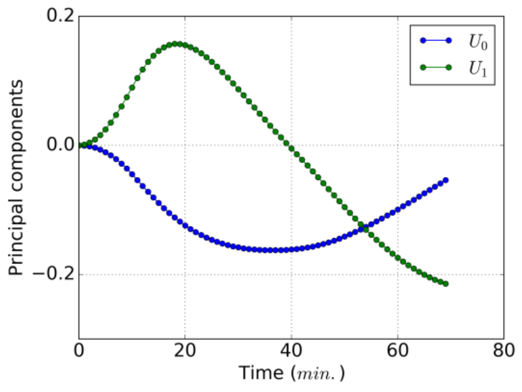
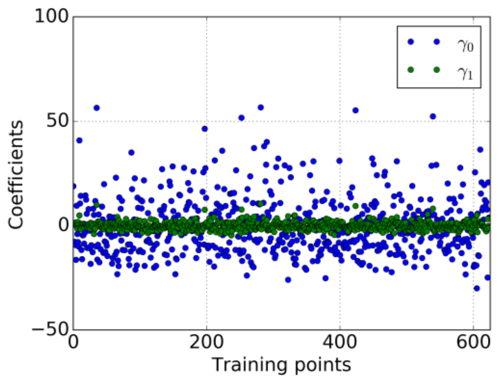


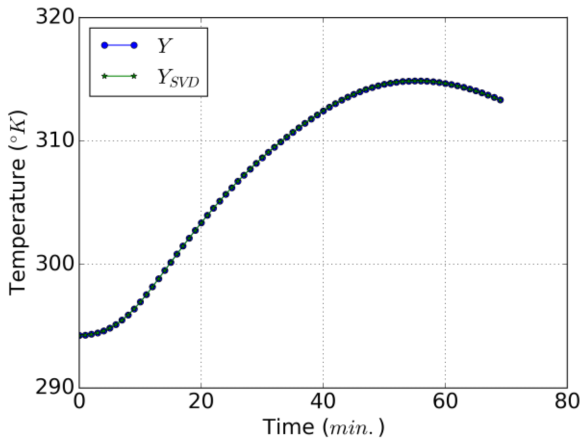
Figure 5.11 Example result of FEA model (@ t = 1800s)



(a) principal components



(b) coefficients



(c) fitting by SVD

Figure 5.12 SVD decomposition example (@$\boldsymbol{d}^{(0)}$)

We build surrogate models for each of the 2 coefficients, $\gamma_0\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), \gamma_2\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), i = 0, 1, \cdots, 399$, and the inputs are the 4 neighboring ks. Thus the total number of surrogate models will be $800 = 2 \times 400$. The training of those surrogate models can be parallelized by following the procedure in Sec. 5.5.3. Fig. 5.12 shows the performance of the trained surrogate model. Here 80% of the data (500 data points) are used for training, while 20% of the data are used for validation.

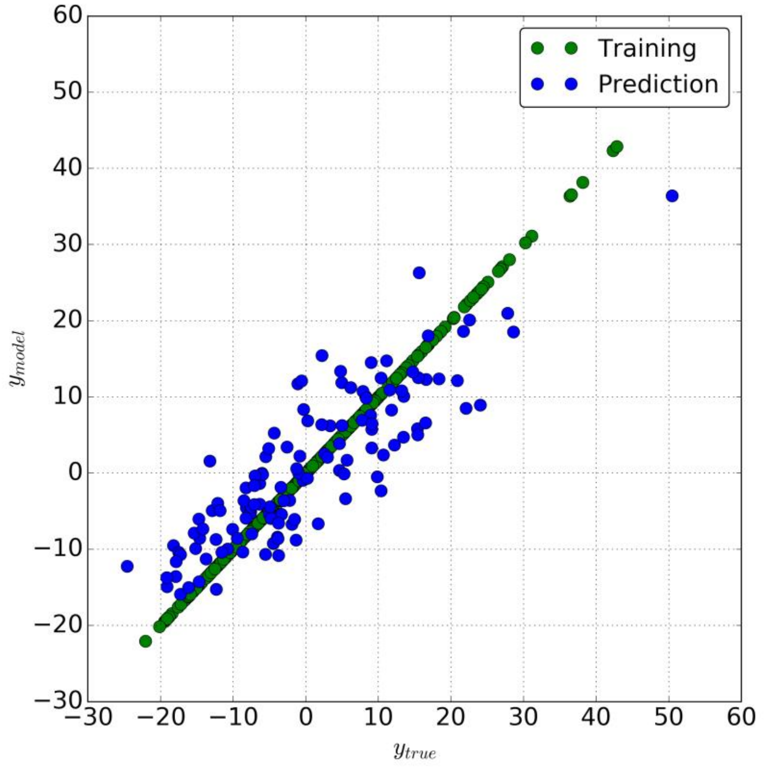


Figure 5.13 Performance of surrogate model

5.7.4 Calibration

After all the surrogate models are trained for each spatial location, the calibration variables, model outputs, and observations can be represented using a Bayesian network. Fig. 5.14 sows the network for one location (4 blocks), in which ellipses are random variables, and squares are observations. Red ellipses denote the random variables that represent the conductivity coefficients $k_j, j = 0, 1, \cdots, 440$ to be calibrated, while the yellow ellipses denote random variables that represent the SVD coefficients of model outputs for each spatial location $\gamma_0\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), \gamma_2\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), i = 0, 1, \cdots, 399$, which can be obtained from the corresponding surrogate model. Each blue ellipse represents the temperature random variable T_i for a spatial location i, where $i = 0, 1, \cdots, 399$. Note here that each T_i follows a multivariate normal distribution $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = E\left[T_{i,l}\right], l = 0, 1, \cdots, 69$ and $\boldsymbol{\Sigma} = Cov\left[T_{i,l}, T_{i,m}\right], l = 0, 1, \cdots, 69;\ m = 0, 1, \cdots, 69$. Here E refers to the expectation function and Cov refers to the covariance function. In our case,

$$\boldsymbol{\mu}_{\boldsymbol{y}_{obs}}\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, t\right) \approx \boldsymbol{\mu}_i(t) + \boldsymbol{\mu}_{\widehat{\gamma}_0}\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t\right) + \boldsymbol{\mu}_{\widehat{\gamma}_1}\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t\right) + \mu_\delta(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t}) \quad (5.18)$$

$$\boldsymbol{\Sigma}_i(j,k) = \sigma^2_{\widehat{\gamma}_0}\left(\boldsymbol{d}^{(i)}, \hat{\beta}\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t_j\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t_k\right) + \sigma^2_{\widehat{\gamma}_1}\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t_j\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t_k\right), \forall j, k = 0, 1, \cdots, 69 \quad (5.19)$$

Given the observation $\{T_{i,l}\}$, where $i = 0, 1, \cdots, 399$, and $l = 0, 1, \cdots, 69$, the likelihood function computation is parallelized following the procedure in Sec.5.6.4. Finally, thermal conductivity coefficients at each spatial location are updated via the parallelization of MCMC.

Fig. 5.15 shows two examples of the calibration results, where the prior and posterior of parameters and are plotted. It is observed reduction of variance happens for both. Similar performance was observed for all the other parameters. Fig. 5.17 shows an overview of all the 441 calibrated parameters, where mean values are plotted. Besides this, it will be useful to check the correlations among the calibrated parameters. For example, the correlations between parameter k_{210} and the other parameters in the same row $k_{211}, k_{212}, \cdots, k_{230}$ (Fig. 5.4) are calculated and plotted in Fig. 5.16. A general decreasing trend can be observed except one outlier (marked in red).

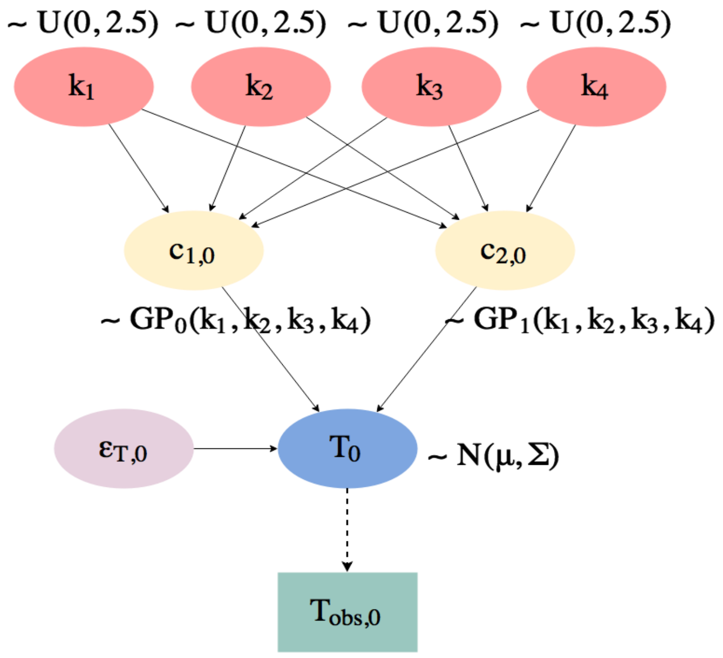
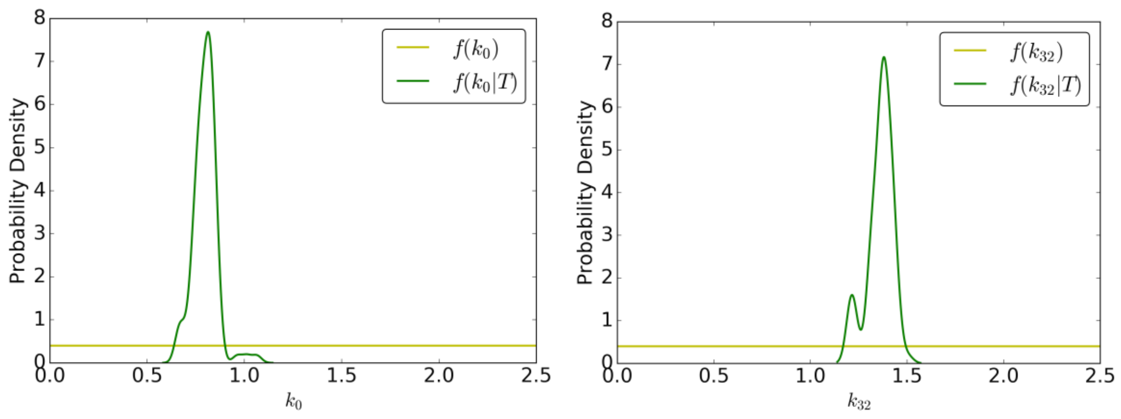


Figure 5.14 Bayesian network for calibration at location 0 (See Fig. 5.4)



(a) k_0 (b) k_{32}

Figure 5.15 Calibration results

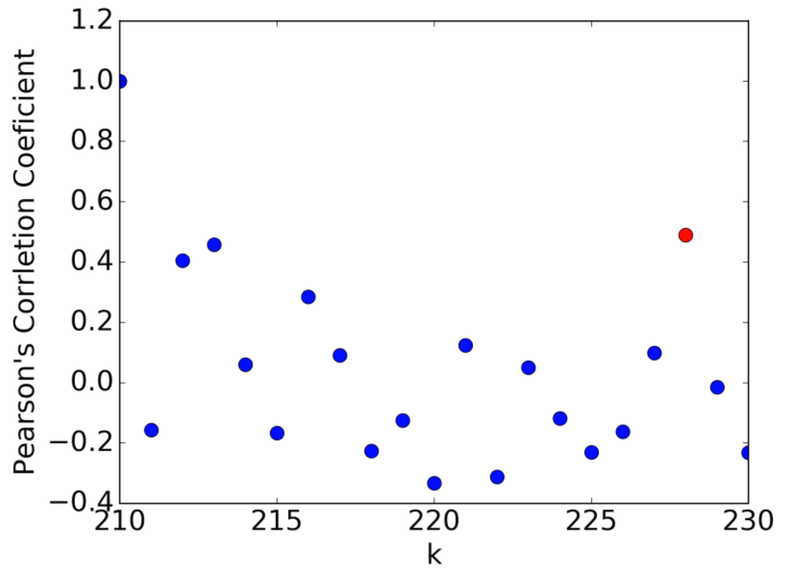


Figure 5.16 Correlation of k_{210} with the other nodes in the same row

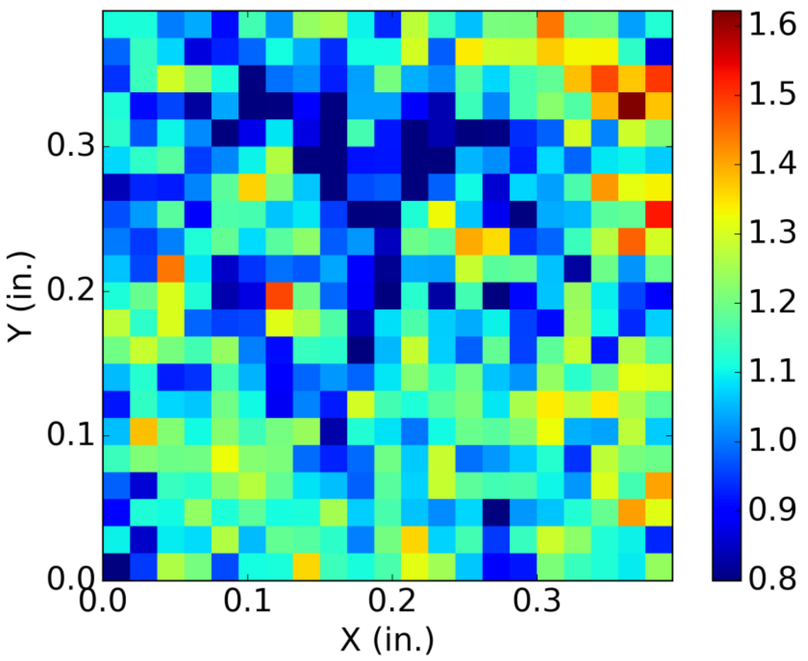


Figure 5.17 Calibration result shown over the slab top surface (mean)

5.7.5 MapReduce Performance

Now we discuss the performance of MapReduce in FEA parallelization. In this study, 50 nodes were used for parallelization. For the purpose of comparison, computation using the traditional sequential method at a single node was also performed. The configurations of computers are shown in Table 5.2.

Table 5.2 Nodes comparison

Method	**CPU (GHZ)**	**Memory (GB)**
Desktop	3.4×8	12
Cluster node	2.3	10

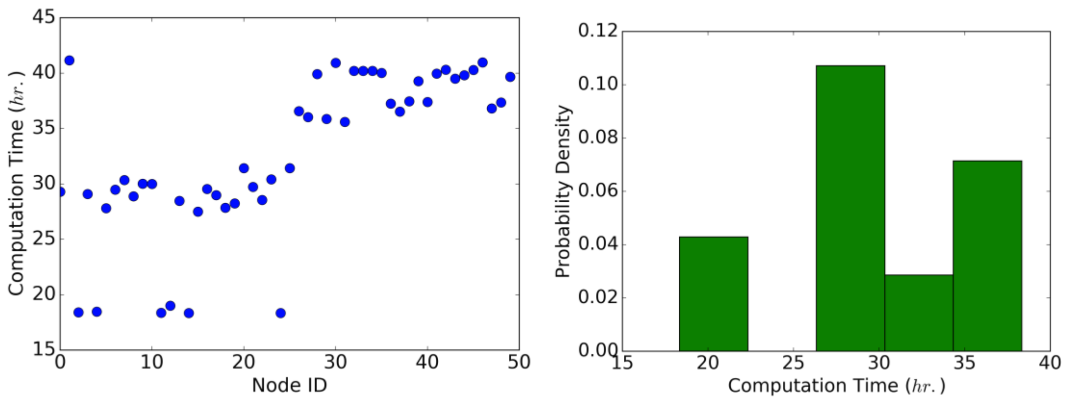
Table 5.3 Time cost of traditional method and MapReduce method

Method	**Time (hr.)**
Desktop	363
Cluster node (50 nodes)	42

It is worth noting that the CPU clock speed and memory size of the local machine where the traditional methods were running are larger than that of the cluster nodes. The comparison between the time cost of the traditional method and MapReduce method is shown in Table 5.3. MapReduce showed significant computational efficiency (almost 9 times faster). It can be expected that as the number of nodes increases, the time cost could reduce further, but may not be in a linear trend. The reason is that the communication between the master node and slave node also consumes time. The individual time cost shows that model calibration consumes most of the computational resources (Table 5.4). Also the heterogeneity of performance of the cluster nodes is shown through a scatter plot and a histogram in Fig. 5.18.

Table 5.4 Time cost of individual steps on desktop

Method	Time (hr.)
FEA model	6
Surrogate model training	1
Calibration	363



(a) scatter plot (b) histogram

Figure 5.18 Computational nodes performance

5.8 Summary

This chapter investigated the MapReduce technique to parallelize the model calibration process in a high-dimensional parameter space and in the presence of big data, in order to make the computation efficient without lowering the accuracy. MapReduce is investigated in three steps of the model calibration process: (1) multiple runs of the original physics model to generate training points to build an inexpensive surrogate model, (2) training of the surrogate model to be used in calibration, (3) construction of likelihood functions for large volume observations, and Bayesian posterior construction (via the MCMC algorithm) using the surrogate model and likelihood function. The methodology is illustrated for the estimation of heterogeneous thermal conductivity at different locations in a damaged concrete structure, using data from infrared thermography (IR).

Future research needs to address several extensions. First of all, a single surrogate model could be created, instead of multiple small-size surrogate models. This will be investigated in next Chapter. In that case, spatial correlation can be handled with no approximation. However, since there would be only one model, the training process will be parallelized internally, instead of doing the parallelization file-wise. Furthermore, since the single surrogate model will have large number of parameters, due to high dimensionality, much more training points are needed. Thus there is tradeoff between multiple small-size surrogate models and a single large surrogate model, which is accuracy vs. effort. On the other hand, when preparing the surrogate model training points, repeated FEA model runs are needed, which were also parallelized externally (file-wise). The FEA node can be parallelized internally, since external parallelization is limited by the number of available licenses of the commercial software. The internal parallelization of FEA model runs may already be available in existing commercial software, which may be taken advantage to save licenses cost. Compared with the random field approach (Nath et al. (2017)), the proposed method is more expensive, but necessary when the material is highly heterogeneous and the structure is damaged, where a random field approach may not be applicable.

CHAPTER 6

BIG DATA ANALYTICS IN DISTRIBUTON SURROGATE MODELING

6.1 Background

As reviewed in Sec. 2.5, there are two types of surrogate models: response surrogate and distribution surrogate. As a response surrogate, the Gaussian process surrogate model is used in Chapter 4 and Chapter 5. For a high dimensional problem, when using response surrogate, multiple surrogate models are required for a field output. In contrast, a single evaluation of the distribution surrogate provides the entire output distribution considering all the uncertain variables at a given value of the input variable (Liang, 2015). In this chapter, we will address the big data analytics in distribution surrogate modeling.

A significant benefit of the distribution surrogate is the ability to consider spatial variability of heterogeneous properties in one single model, instead of multiple smaller-dimensional models as was considered in Chapter 5. In addition, inference with approximate distribution surrogates such as a Gaussian mixture model (McLachlan, 2000; Bishop, 2006) is much faster, since analytical solutions are available, which can be directly used to obtain the conditional distribution (in model calibration or prediction).

However, when the data size is large and parameter dimension is high, the training of the Gaussian mixture model (GMM) becomes expensive, thus posing a challenge to traditional computing (sequential computing). Therefore this chapter focuses on the parallelization of GMM training, including the data processing (which is used for the observation) and model training (parallelize the training process). Since after surrogate model is trained, analytical solutions can be obtained for the posterior distributions, there is no need for the calibration calculations to be parallelized.

Different schemes of scalable GMM have been investigated by researchers. Feldman et al. (2011) proposed a way of constructing core-sets (i.e., weighted subsets of the data) for mixtures of Gaussians to allow the GMM to be applicable for a massive data set. It was found that Gaussian mixtures admit core-sets whose size is independent of data size. Jin et al. (2005) proposed scalable GMM based on data summarization. Parsimonious GMM (McNicholas et al., 2009) is another data reduction method combined with parallelization, which accelerates model training and selection. Both of the three methods above are achieved by approximations. On the other hand, researchers studied the parallelization without data reduction. For example, Kumar et al. (2009) proposed parallelization of GMM via CUDA (Compute Unified Device Architecture) on GPUs. However, this will face the limitations of GPU parallelization discussed earlier (Sec. 5.1). Kwedlo (2014) implemented GMM parallelization using MPI, which is a shared memory parallelization based on data decomposition. This will inherit the limitations of MPI (Sec. 3.1).

In this chapter, MapReduce parallelization of GMM will be investigated. Since GMM relies on the Expectation-Maximization (EM) algorithm (described in Chapter 2), the parallelization of E-step and M-step can be realized either by partitioning the samples or by partitioning the components. Furthermore, in order to perform model selection, GMMs with different configurations can also be parallelized. Thus this chapter proposes three different options for parallelization. Since the parallelization of E-step and M-step is performed inside a single GMM, we denote this form as ‘internal’ GMM parallelization; on the other hand, the parallelization at the model selection level is termed 'external' GMM parallelization in the discussion below.

6.2 Challenges due to High-Dimensional Model Parameters

Due to material variability, the material parameters to be calibrated, $\boldsymbol{\theta}$, may be defined as functions of locations $\boldsymbol{d}$. Since $\boldsymbol{d}$ is a large vector, the number of parameters $\boldsymbol{\theta}(\boldsymbol{d})$ is very large. Therefore, the calibration process is unaffordable using the conventional model calibration method, if it is impossible to develop a parametric representation of the model parameter over the spatial domain. Furthermore, full field observations (such as optical or thermal images) collected over a long time period will bring in the big data issue. Thus this section discusses our approach to handle the spatially varying parameters and temporal correlation.

6.2.1 Spatially Varying Parameters

As explained in Sec. 5.5.1, to build a surrogate model for the high-dimensional spatially varying response as a function of the calibration parameters, we first classify the calibration parameters into two categories: spatially constant calibration parameters ($\boldsymbol{\theta}^c$) and spatially varying calibration parameters ($\boldsymbol{\theta}^s(\boldsymbol{d})$). In this chapter, in order to build a full-scale surrogate model, $\boldsymbol{\theta}^s(\boldsymbol{d})$ is directly used as inputs, instead of only using the $\boldsymbol{\theta}^s(\boldsymbol{d})$ in the neighboring locations of d as the inputs of $\boldsymbol{y}$ at $\boldsymbol{d}$. The response at location $\boldsymbol{d}^{(i)}$ is represented as

$$\boldsymbol{y}\left(\boldsymbol{d}^{(i)}\right) = \hat{G}(\boldsymbol{\theta}^c, \boldsymbol{\theta}^s(\boldsymbol{d})) \tag{6.1}$$

where $\boldsymbol{d}^{(i)}$ is the i-th spatial coordinate, and $\hat{G}(\cdot)$ is the surrogate model. As mentioned earlier, GMM is used as the surrogate model in this chapter.

6.2.1.1 Generation of Training Points

Defining $\boldsymbol{\beta} = [\boldsymbol{\theta}^c, \boldsymbol{\theta}^s(\boldsymbol{d}^{(1)}), \boldsymbol{\theta}^s(\boldsymbol{d}^{(2)}), \cdots,, \boldsymbol{\theta}^s(\boldsymbol{d}^{(m)})]$, we first generate n_s training points for $\boldsymbol{\beta}$. For each training point $\boldsymbol{\beta}^{(i)}$, the response field is obtained using the original simulation model G as below:

$$\boldsymbol{y}(\boldsymbol{d}^{(j)}, \boldsymbol{\beta}^{(i)}) = G(\boldsymbol{\beta}^{(i)}) \tag{6.2}$$

where $\boldsymbol{y}(\boldsymbol{d}^{(j)}, \boldsymbol{\beta}^{(i)})$ denotes the response at the j-th spatial location of the i-th training point $\boldsymbol{\beta}^{(i)}$. It should be noted that the output is a field response (as indicated in Eq. (6.2)) for given training point $\boldsymbol{\beta}^{(i)}$.

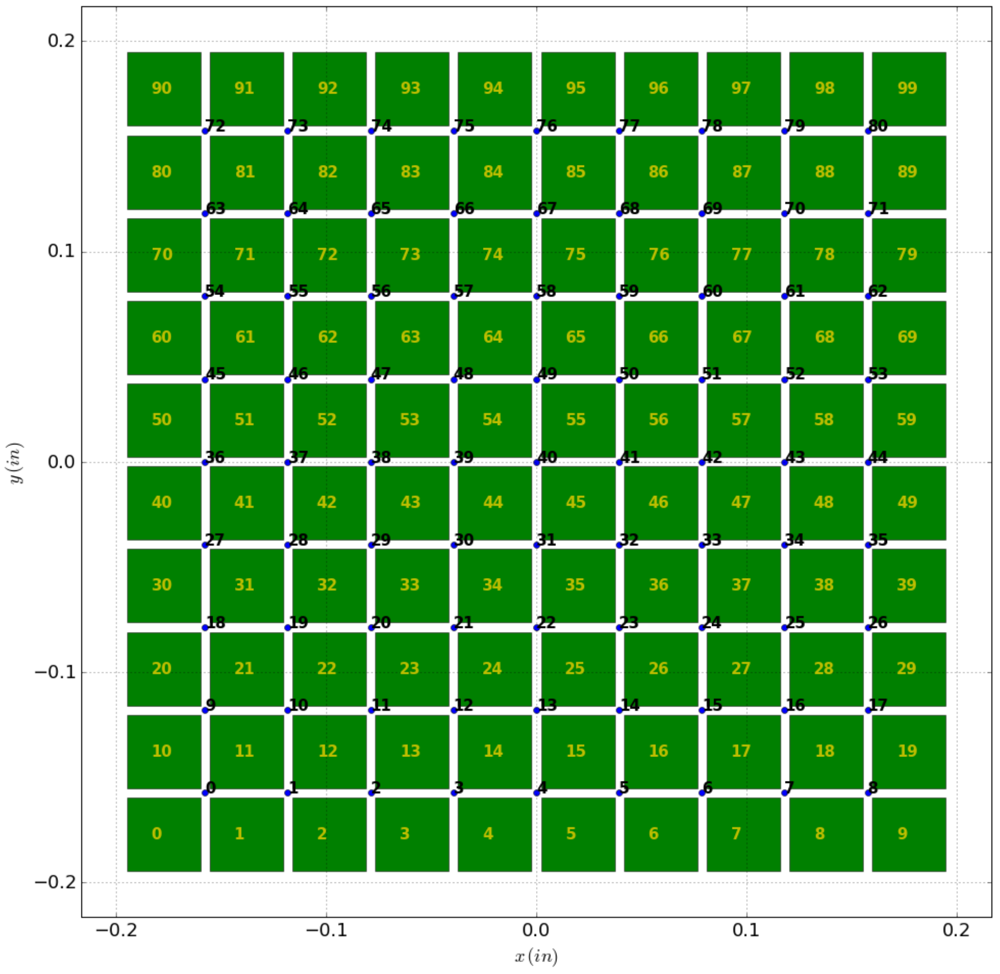


Figure 6.1 10 × 10 calibration grid and 9 × 9 observation points

After performing simulations at all the training points, a data matrix is obtained as $\boldsymbol{y}_{total} = \{\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}), i = 1, \cdots, m; j = 1, \cdots, n_s\}$. Note that the above matrix is obtained by reorganizing the spatial response into a one-dimensional array (i.e., $\boldsymbol{y}(\boldsymbol{d}, \boldsymbol{\beta}^{(j)}) = \{\boldsymbol{y}(\boldsymbol{d}^{(1)}, \boldsymbol{\beta}^{(j)}), \boldsymbol{y}(\boldsymbol{d}^{(2)}, \boldsymbol{\beta}^{(j)}), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(m)}, \boldsymbol{\beta}^{(j)})\}$ denotes the responses at all the spatial locations).

6.2.1.2 Time History Output

With the training data matrix $\boldsymbol{y}_{total}$, we try to build the surrogate model. We extract the input training points as $\widehat{\boldsymbol{\beta}}_{in} = \widehat{\boldsymbol{\beta}}$, and the corresponding output training points as $\boldsymbol{y}_{out} = \boldsymbol{y}(\boldsymbol{d}, \widehat{\boldsymbol{\beta}})$. However, when the simulation model is performed over time, $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ is a time-dependent trajectory even for a specific spatial location $\boldsymbol{d}^{(i)}$ and we have $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) = [\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_1), \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_2), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_{n_t})]$, where n_t is the number of time instants. This introduces extra challenge to the surrogate model construction. Next, we investigate how to address this issue using singular value decomposition (SVD).

6.2.2 Handling Temporal Correlation

Singular value decomposition (SVD) is a multivariate statistical method to describe a large amount of high-dimensional data be mapping to a low-dimensional space (Chatterjee 2000). SVD can be used for handling the temporal correlation of the response. Given m data points over the spatial domain $\boldsymbol{\Omega}$ for n_t time domain realizations, a data matrix can be collected as follows:

$$\boldsymbol{\omega} = [\boldsymbol{\omega}(\xi_1), \boldsymbol{\omega}(\xi_2), \cdots, \boldsymbol{\omega}(\xi_s)]^T = \begin{bmatrix} \omega(t_1,\xi_1) & \omega(t_1,\xi_2) & \cdots & \omega(t_1,\xi_{n_t}) \\ \omega(t_2,\xi_1) & \omega(t_2,\xi_2) & \cdots & \omega(t_2,\xi_{n_t}) \\ \vdots & \vdots & \ddots & \vdots \\ \omega(t_{n_t},\xi_1) & \omega(t_{n_t},\xi_2) & \cdots & \omega(t_{n_t},\xi_{n_t}) \end{bmatrix}^T \quad (6.3)$$

where $\boldsymbol{\omega}(\xi_i) = [\omega(t_1,\xi_i), \omega(t_2,\xi_i), \cdots, \omega(t_{n_t},\xi_i)]$ is the i-th realization.

This large amount of high-dimensional data can be mapped to a low-dimensional representation by using SVD as $\boldsymbol{\omega} = \boldsymbol{VMU}^T$, where $\boldsymbol{V}$ is a $s \times n_t$ matrix, $\boldsymbol{U}$ is a $n_t \times n_t$ orthogonal matrix and $\boldsymbol{M}$ is a $n_t \times n_t$ rectangular diagonal matrix with non-negative real numbers $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_m]$ on the diagonal. Here we donate $\boldsymbol{\gamma} = \boldsymbol{VM}$, and the matrix can be constructed as

$$\boldsymbol{\omega}(\cdot,\xi_i)^T \approx \sum_{j=1}^{r} \gamma_{ij} \boldsymbol{U}_j \quad (6.4)$$

where $\boldsymbol{\omega}(\cdot,\xi_i)^T$ is the i-th row of $\boldsymbol{\omega}$, γ_{ij} is the element of $\boldsymbol{\gamma}$ at i-th row and j-th column, $\boldsymbol{U}_j$ is the j-th important feature vector used to approximate $\boldsymbol{\omega}$, and r is the number of important features used. The number of features r is determined based on the magnitudes of the singular values $\boldsymbol{\lambda}$ (Xu, 1998).

Based on SVD, the response at spatial location $\boldsymbol{y}_{all}^{(i)} = \{\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k), j = 1, 2, \cdots, s, k = 1, 2, \cdots, n_t\}$ is re-constructed as

$$\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k) \approx \boldsymbol{\mu}_i(t_k) + \sum_{q=1}^{r} \gamma_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) U_q(\boldsymbol{d}^{(i)}, \boldsymbol{t}_k), \forall j = 1, 2, \cdots, s; k = 1, 2, \cdots, n_t \quad (6.5)$$

where $\boldsymbol{\mu}_i(t_k)$ is the mean value at location $\boldsymbol{d}^{(i)}$ at time instant t_k, $\gamma_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ is the q-th latent response of spatial location $\boldsymbol{d}^{(i)}$ for the j-th training point, and $\boldsymbol{U}_q(\boldsymbol{d}^{(i)}, t_k)$ is the value of the q-th important feature $\boldsymbol{U}_q$ of $\boldsymbol{d}^{(i)}$ at time instant t_k.

Eq. (6.5) shows that the variation in the high-dimensional response mainly comes from the variation in $\boldsymbol{\gamma}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) = [\gamma_1(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)}), \gamma_2(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)}), \cdots, \gamma_r(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(i)})]$, which denotes the value of $\boldsymbol{\gamma}$ of the response at $\boldsymbol{d}^{(i)}$ for the j-th training point. The dimension of $\boldsymbol{\gamma}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ is usually much smaller than that of the response $\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) = [\boldsymbol{y}(\boldsymbol{d}^{(i)}, t_1), \boldsymbol{y}(\boldsymbol{d}^{(i)}, t_2), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(i)}, t_{n_t})]$.

With the training points $\boldsymbol{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}), \forall q = 1, 2, \cdots, r;\ j = 1, 2, \cdots, s$ and $\widehat{\boldsymbol{\beta}}_{in}^{(i)} = [\widehat{\boldsymbol{\beta}}_1^{(i)}, \widehat{\boldsymbol{\beta}}_2^{(i)}, \cdots, \widehat{\boldsymbol{\beta}}_s^{(i)}]$, we construct surrogate model for $\boldsymbol{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}), \forall q = 1, 2, \cdots, r$. After substitute $\boldsymbol{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ with surrogate model $\boldsymbol{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$, Eq. (6.5) becomes:

$$\boldsymbol{y}(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}, t_k) \approx \boldsymbol{\mu}_{\boldsymbol{i}}(t_k) + \sum_{q=1}^{r} \hat{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)}) U_q(\boldsymbol{d}^{(i)}, t_k), \forall j = 1, 2, \cdots, s; k = 1, 2, \cdots, n_t \qquad (6.6)$$

in which $\widehat{\boldsymbol{\gamma}}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$ stands for the q-th surrogate model associated with the spatial location $\boldsymbol{d}^{(i)}$. Note that $\gamma_1, \gamma_2, \cdots, \gamma_r$ are not the original response but latent response obtained through SVD.

6.3 Construction of Gaussian Mixture Model

From the discuss above, after repeated runs of FEA models with different parameters, training points of $\{(\boldsymbol{\beta}^{(j)}, \boldsymbol{y}^{(j)}), \forall\, j = 1, 2, \ldots, n_s\}$ can be obtained, where $\boldsymbol{y}(\boldsymbol{d}, \boldsymbol{\beta}^{(j)}) = \{\boldsymbol{y}(\boldsymbol{d}^{(1)}, \boldsymbol{\beta}^{(j)}), \boldsymbol{y}(\boldsymbol{d}^{(2)}, \boldsymbol{\beta}^{(j)}), \cdots, \boldsymbol{y}(\boldsymbol{d}^{(m)}, \boldsymbol{\beta}^{(j)})\}$ is the output for parameter $\boldsymbol{\beta}^{(j)}$. Note the $\boldsymbol{y}(\boldsymbol{d}, \boldsymbol{\beta}^{(j)})$ above is a function of time and space, thus to handle the spatial correlation, SVD method will be used (Eq. (6.6)) to obtain r set of basis U_q, and coefficients $\hat{\gamma}_q(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}^{(j)})$. Now GMM can be trained via the details discussed in Sec. 2.5.2. Note that the number of components (K in Eq. (2.7)) of a GMM is undetermined. Determine the number of components is a very important step, which will be discussed below.

Selecting the number of components in a GMM is a process of selecting the model with the least information loss. There are two commonly used criteria in model selection – the Akaike information criterion (Akaike, 1974) and the Bayesian information criterion (Schwarz, 1978). In this chapter, Akaike information criterion is applied for model selection. The AIC is defined as

$$AIC = 2k - 2\ln(L) \tag{6.7}$$

where L is the maximized value of the likelihood function of the model G, i.e., $L = p(\boldsymbol{x}|\boldsymbol{\theta}, G)$, where $\boldsymbol{\theta}$ are the parameter values that maximize the likelihood function; $\boldsymbol{x}$ is the observed data; n is the number of data points in $\boldsymbol{x}$; and k is the number of free parameters to be estimated.

In order to find the model with the lowest AIC score, a global optimization method (such as genetic algorithm) could be applied. However, if the potential model numbers is not large, we can enumerate all the model configurations (number of components, i.e., N in Eq. (6.2)).

6.4 Parallelization of GMM Construction

As mentioned in Sec. 6.1, three options for parallelization of GMM construction are developed here. The first option is to solve all the candidate GMM models (i.e., with different numbers of components) in parallel, but within each model solving process (i.e., E-step and M-step), it is sequential (scheme 1). We refer to this type of parallelization as “external”. The second option is to parallelize the E-step and M-step for each of the GMM model. Inside each model, the E-step and M-step are solved in parallel, but different models are solved sequentially (scheme 2 and scheme 3). We refer to this type of parallelization as “internal”. Since potentially both big data property (large sample size N) and high-dimension property (large number of components required K) are present, the parallelization can be done either by partitioning the training samples (scheme 2) or by partitioning the components (scheme 3).

6.4.1 GMM Parallelization Scheme 1: MapReduce Implementation of GMM Model Selection (external parallelization)

It is straightforward to parallelize the GMM model selection, since each individual model training is independent of the others; therefore it is naturally reasonable to split all the n model training tasks into M partitions, which will be sent to m nodes (Fig. 6.1 (b)). Similar to the parallelization of data processing, M should be greater than or equal to m to avoid waste of resources, since otherwise there would be idle workers.

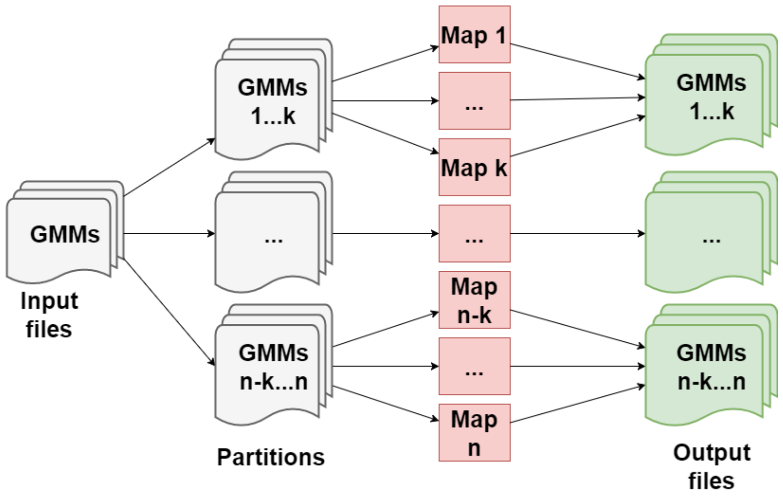
Pseudocode 6.1 in Fig. 6.2 (a) is the MapReduce pseudocode and the schematic description is shown in Fig. 6.2 (b). 'GMMs' refers to the GMM training tasks, which is composed by GMM models with different component numbers. 'M' is the number of partitions as explained above. 'count()' is used for triggering the parallelization, as usual. The 'mapper' function is the function to be executed by workers. The argument 'x' is the task id received by a worker, which will be used for reading the input data (realized by 'ReadData(x)' function). 'EM' is the function of performing the training process, with the trained model ('GMM_x') saved on to the disk via 'WriteData(GMM_x)' function.

```
Pseudocode 6.1:

function mapper(x):
    InputData = ReadData(x)
    GMM_x = EM(InputData)
    WriteData(GMM_x)
    return (x, 0)

SparkContext(appName="myApp").parallelize(GMMs,
M).map(mapper).count()
```

(a) MapReduce pseudocode



(b) Schematic description

Figure 6.2 GMM parallelization scheme 1

6.4.2 GMM Parallelization Scheme 2: MapReduce Implementation of EM by Partitioning the Samples (Internal Parallelization)

For this parallelization scheme, in the E-step, N training points (samples) are split into M partitions; and for each sub-group of training points ($\boldsymbol{x}_i$), log likelihood will be calculated, which will be combined and used for the calculation of posterior distributions (Eq. (6.8)). In

M-step, similarly, each node will process the subset $\boldsymbol{x}_i$ and after combination, $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\pi}_j$ can be obtained (Eqs. (2.9) – (2.12)). However, since in the M-step, the most computationally intense part is the calculation of covariance matrix, it is practical to only parallelize the computation of $\boldsymbol{\Sigma}_{\mathrm{j}}$.

Pseudocode 6.2 in Fig. 6.3 (a) is the MapReduce pseudocode of GMM parallelization scheme 2. Similar to Pseudocode 6.1, 'GMMs' refers to the GMM training tasks (total number is n), which is composed by GMM models with different component numbers. The outer loop 'for GMM in GMMs' indicates that each GMM with a certain configuration (number of components) will be trained one by one (sequentially). For the GMM being trained, 'ReadData' function will read training samples (stored in variable 'samples'), as well as configurations (stored in variable 'num_of_components'). Then, 'InitializeComponents' is used for initialization of components ($\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\pi}_j$). The second loop 'for component in components' indicates the iteration over components, for each component which is being worked on, E-step and M-step will be parallelized inside 'function E_step and function M_step'. Inside 'function E_step', as usual, 'count()' is the function to trigger the parallelization, which splits 'samples' into M parts, and sends to the available workers (slave nodes). After likelihood is calculated for the assigned samples partition, it is save onto disk by 'WriteData'. 'CombineLikelihood' is used to merge all calculated likelihood values by adding the log likelihood values. 'ComputePosterior' will compute the posterior following Eq. (2.8). Similar procedure is in 'function M_step', inside which covariance matrix is solved in parallel due to the reason explained in the previous paragraph.

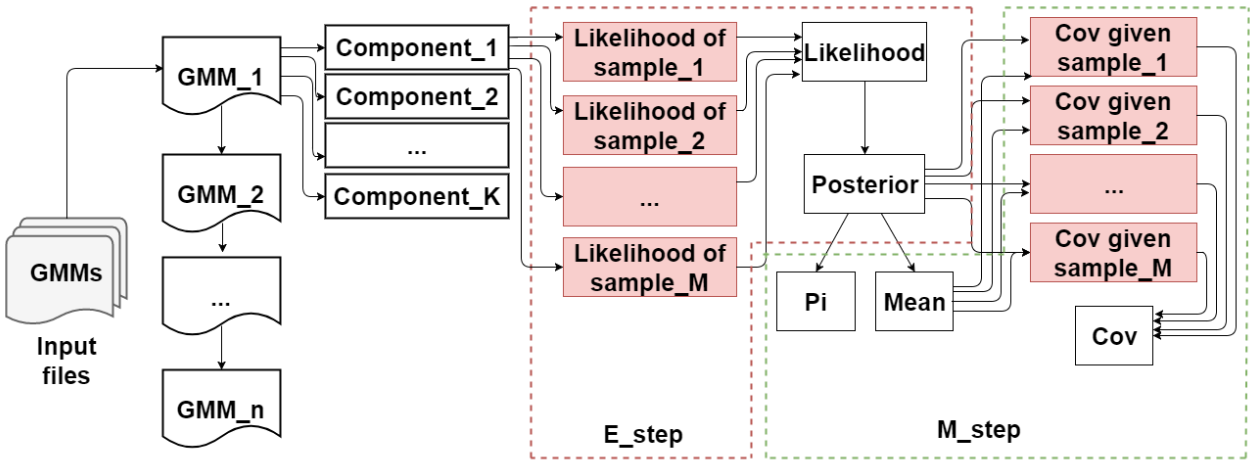
```
Pseudocode 6.2:

for GMM in GMMs:
    samples, num_of_components = ReadData(GMM)
    components = InitializeComponents (num_of_components)

    for component in components:
        function E_step (samples):
            function mapper(x):
                Likelihood_x = LikelihoodCal(x)
                WriteData(Likelihood_x)
                return (x, 0)
            SparkContext(appName="myApp").parallelize(samples, M).map(mapper).count()
            CombineLikelihoods()
            posterior = ComputePosterior()

        function M_step (samples, posterior):
            pi = ComputeMixingcoefficients(samples)
            mean = ComputeMean(Samples)
            function mapper(x):
                cov_x = ComputeCov(x, mean)
                WriteData(pi, mean, cov)
                return (x, 0)
        SparkContext(appName="myApp").parallelize(samples, M).map(mapper).count()
        cov = CombineCov()
```

(a) MapReduce pseudocode



(b) Schematic description (red boxes are map tasks)

Figure 6.3 GMM parallelization scheme 2

6.4.3 GMM Parallelization Scheme 3: MapReduce Implementation of EM by Partitioning the Components (Internal Parallelization)

For this parallelization scheme, in the E-step, K components are split into M partitions, for each sub-group of components ($\boldsymbol{k}_i$), log likelihood will be calculated, which will be combined and to be used for the calculation of posterior distributions (Eq. (2.8)). In the M-step, similarly, each node will process the subset $\boldsymbol{k}_i$ and after combination, $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\pi}_j$ can be obtained (Eqs. (2.9) – (2.12)).

Pseudocode 6.3 in Fig. 6.4 (a) is the MapReduce pseudocode of GMM parallelization scheme 3. As the same as Pseudocode 6.2, for each GMM, 'samples' and 'components' will be read and initialized. Then within 'function E_step', likelihood values are calculated in parallel and combined by 'CombineLikelihoods()', and posterior can be obtained based on the combined likelihood. Within 'function M_step', after directly compute $\boldsymbol{\mu}_j$ by

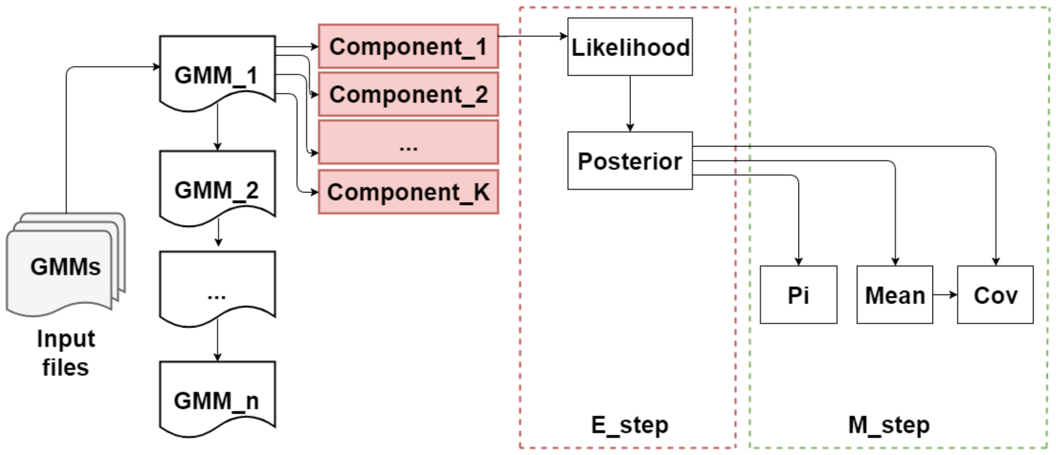
‘ComputeMixingcoefficients’ and $\boldsymbol{\pi}_j$ by ‘ComputeMean’, $\boldsymbol{\Sigma}_\mathrm{j}$ is computed by splitting components, sending to slave nodes by ‘SparkContext’, and combining the partial results by ‘CombineCov()’.

```
Pseudocode 6.3:

for GMM in GMMs:
    samples, num_of_components = ReadData(GMM)
    components = InitializeComponents (num_of_components)
    function E_step (samples, components):
         function mapper(x):
             Likelihood_x = LikelihoodCal(x);
             WriteData(Likelihood_x);
             return (x, 0)
         SparkContext(appName="myApp").parallelize(components, M).map(mapper).count()
         CombineLikelihoods()
         posterior = ComputePosterior()

    function M_step (samples, components, posterior):
        pi = ComputeMixingcoefficients(samples)
        mean = ComputeMean(samples)
        function mapper(x):
            cov_x = ComputeCov(x, mean)
            WriteData(pi, mean, cov)
            return (x, 0)
        SparkContext(appName="myApp").parallelize(components, M).map(mapper).count()
        cov = CombineCov()
```

(a) MapReduce pseudocode



(b) Schematic description (red boxes are map tasks)

Figure 6.4 GMM parallelization scheme 3

6.5 MapReduce for FEA Model Runs

To prepare the training data for GMM models, a large number of FEA model runs is required, which is time consuming. Thus MapReduce implementation of FEA model runs can be used to parallelize the FEA runs, following the procedure described in Sec. 5.4.

6.6 Parallelization of Data Processing

To prepare the observation data for model calibration, data processing can be parallelized using MapReduce following the procedure described in Sec. 3.3.5.

In summary, the steps for the calibration of high-dimensional model parameters using big data analytics are: (1) parallelization of FEA model runs; (2) parallelization of GMM surrogate model training using scheme 1, 2, or 3; (3) parallelization of observation data processing; and (4) model calibration using the GMM surrogate and observation data.

6.7 Numerical Example

6.8.1 Experiment Setup, Data Sampling and Processing

The proposed parallelized Bayesian surrogate modeling for model calibration will be applied on the same concrete slab as in Chapter 5. The experiment setup is explained in Sec. 5.7.1.1, and the data sampling and processing procedures follow the descriptions in Sec. 3.4.2-3.4.5. The same 70 images are used for calibration.

6.8.2 FEA Model

Fig. 6.5 shows the meshed FEA model implemented in commercial software Abaqus, with 7255 nodes and 4078 thermal-coupled elements (128 quadratic brick element and 3950 quadratic tetrahedral elements). The thermal conductivity coefficients at different spatial locations on the top surface need to be calibrated. In the FEA model, the spatial locations are represented as a 10×10 grid as shown in Fig. 6.5. For each calibration block location, the thermal conductivity is considered to be constant. We use $81\ (= 9 \times 9)$ observation points on the top surface.

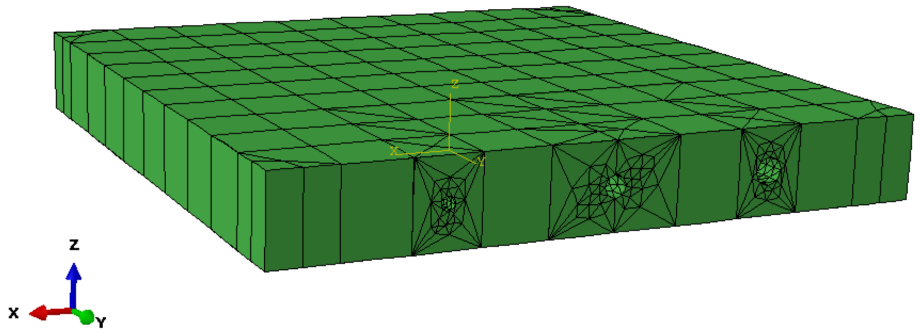


Figure 6.5 FEA model for concrete slab

Except the thermal conductivities which are to be calibrated, the other concrete properties used in this model are the same as in Chapter 5 (Table 5.1). Thermal conductivity is considered to be in the range of $[0.8, 2.5]$ $W \cdot m^{-1} \cdot K^{-1}$. Training points of the surrogate model are obtained using a Latin-hypercube design, and the number of DOE points is set to be $10{,}000$. One example realization of training inputs is shown in Fig. 6.6 (the axis values are block indices in x and y direction). Since in each FEA run, the temperature at all locations can be obtained at the same time, the total number of FEA runs will be $10{,}000$. These $10{,}000$ runs are parallelized via MapReduce as described in Sec. 6.6.

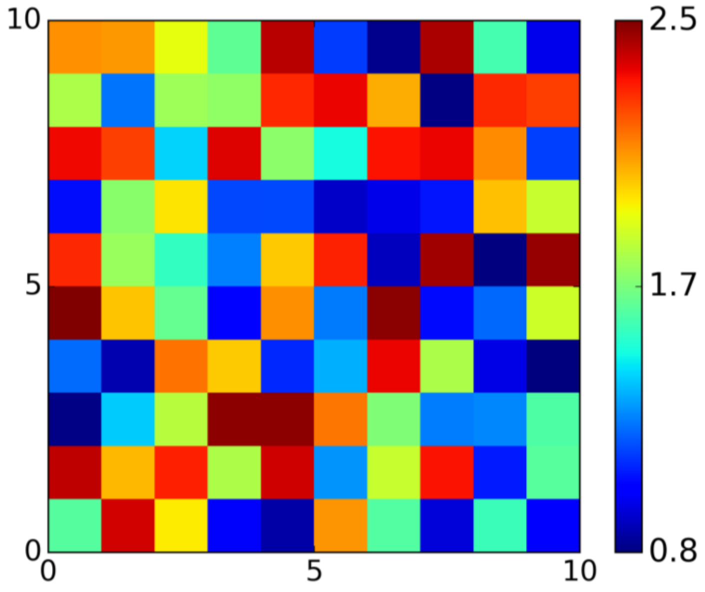


Figure 6.6 Example realization of k values for one training point

6.8.3 Surrogate Model Training

Based on the inputs (conductivity values) and outputs (nodal temperature values in each run), the GMM surrogate model can be obtained. One example output is shown in Fig. 6.7.

For each FEA output, we will have a time series output for 70 time steps (70 mins). For each spatial location i, if we create a surrogate model for each time step, we will lose the correlation between each time step. In order to capture the correlation over time, and also to reduce the dimension, singular value decomposition (SVD) is applied. Following Eq. (6.4), where $\boldsymbol{\omega}$ is the temperature output at each location for all 10000 training points ($10{,}000 \times 70$), $\boldsymbol{V}$ is the left singular vectors (70×70), $\boldsymbol{M}$ is the matrix of singular values (70×70), and $\boldsymbol{U}$ is the matrix of right singular vectors (70×70). Here we choose only the first two components, which means we will use the first two columns of $\boldsymbol{VM}$, and the first two rows of $\boldsymbol{U}$. Thus we have two bases $\boldsymbol{U}_0$ and $\boldsymbol{U}_1$ are used here as an example (Fig. 6.8 (a)), and the corresponding coefficient for each DOE output will have a dimension of 1×2 (Fig. 6.8 (b)). Fig. 6.8 (c) shows that the 2-components SVD captures the temporal history very well.

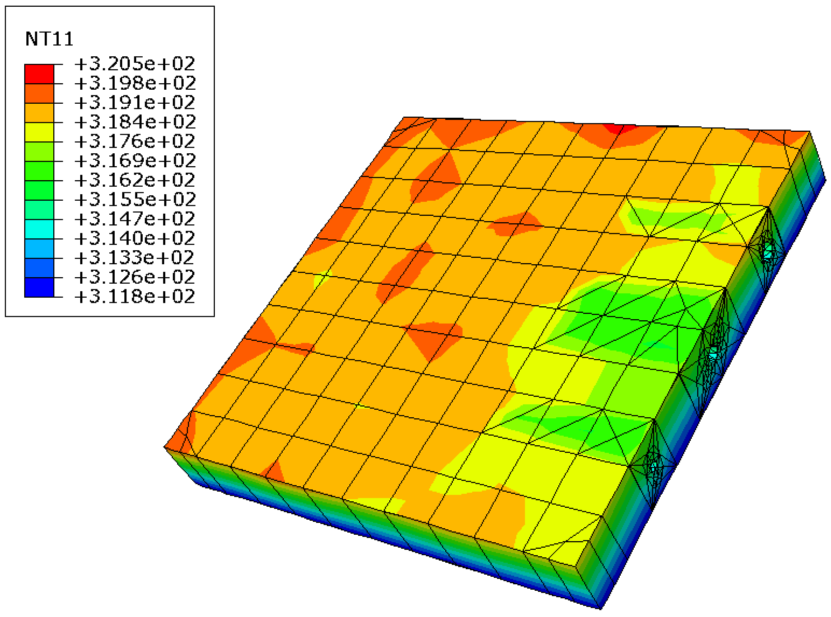
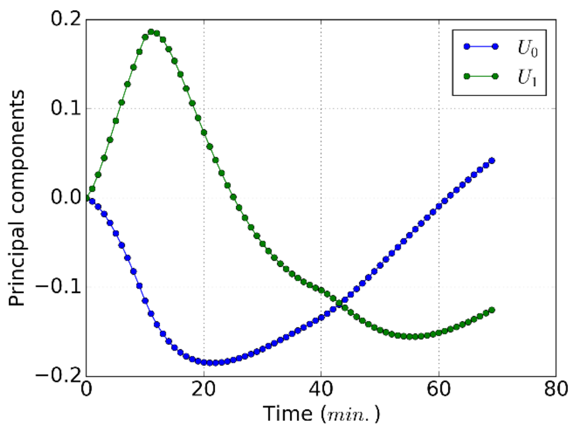
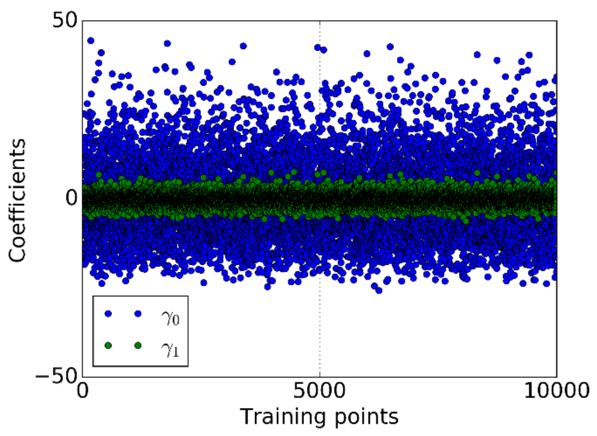


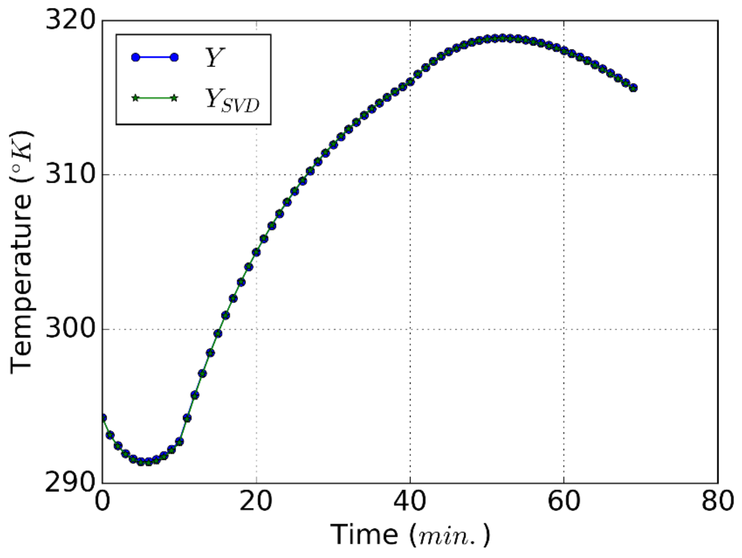
Figure 6.7 Example result of FEA model (@ t = 3000s)



(a) principal components



(b) coefficients



(c) fitting by SVD

Figure 6.8 SVD decomposition example ($@\boldsymbol{d}^{(0)}$)

We build a GMM surrogate model for all coefficients, $\{\gamma_0(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}), \gamma_2(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}), i = 0, 1, \cdots, 80\}$, and the inputs are the 100 ks. Thus the total number of parameters is $282 = 100 + 2 \times 81$. The training of the GMM surrogate model can be parallelized by following the procedure in Sec. 6.5. Fig. 6.9 shows the performance of the trained surrogate model (153 components). Here 80% of the data (8,000 data points) are used for training, while 20% of the data (2,000 data points) are used for validation.

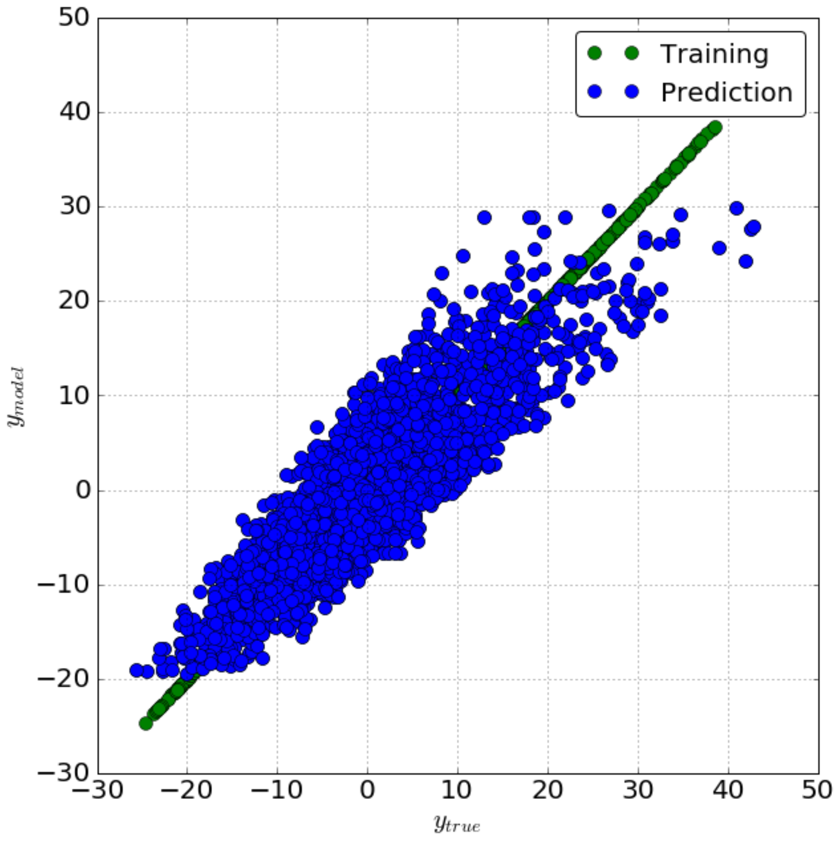


Figure 6.9 Performance of surrogate model

6.8.4 Model Selection

In order to select the optimized model (i.e., the number of GMM components), AIC score is compared among all models with component numbers ranging from $K = 1$ to $K = 500$. The AIC score is plotted in Fig. 6.10, and the GMM model with 153 components is selected.

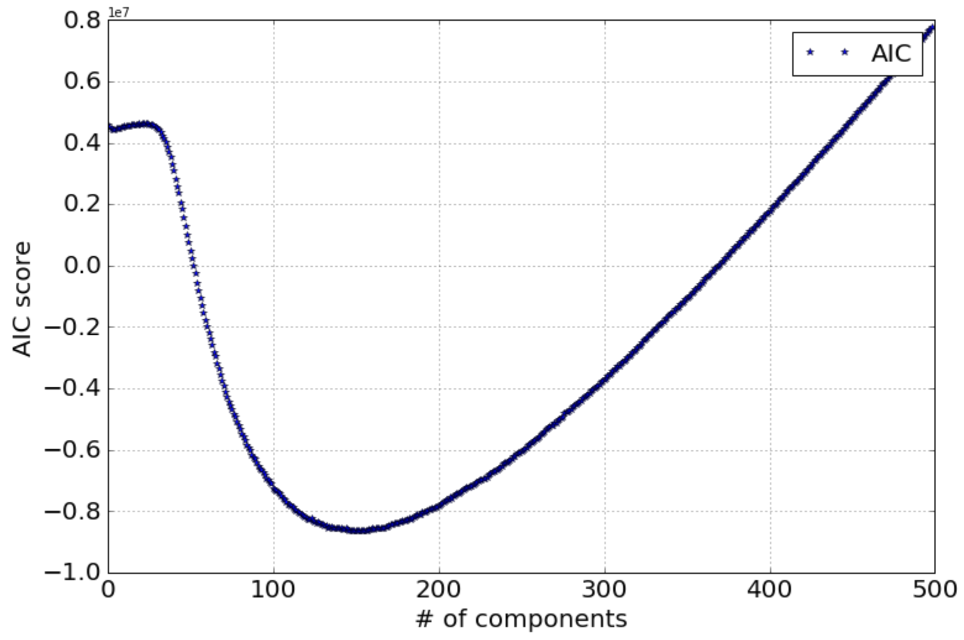


Figure 6.10 Plot of AIC

6.8.5 Calibration

After the GMM surrogate model is trained, the model parameters and observations can be connected using a Bayesian network in Fig 6.11 (ellipses are random variables, and squares are observations). Red ellipses denote the random variables that represent the thermal conductivity coefficients $k_j, j = 0, 1, \cdots, 99$ to be calibrated, while the yellow ellipses denote random variables that represent the SVD coefficients of model outputs for each spatial location $\gamma_0\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), \gamma_2\left(\boldsymbol{d}^{(i)}, \widehat{\boldsymbol{\beta}}\right), i = 0, 1, \cdots, 80$, which can be obtained from the corresponding surrogate model. Each blue ellipse represents the temperature random variable T_i for a spatial location i, where $i = 0, 1, \cdots, 80$. Note here that each T_i follows a multivariate normal distribution $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = E\left[T_{i,l}\right], l = 0, 1, \cdots, 69$ and $\boldsymbol{\Sigma} = Cov\left[T_{i,l}, T_{i,m}\right], l = 0, 1, \cdots, 69;\ m = 0, 1, \cdots, 69.$ Here E refers to the expectation function and Cov refers to the covariance function. In our case,

$$\boldsymbol{\mu}_{y_{obs}}\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, t\right) \approx \boldsymbol{\mu}_i(t) + \boldsymbol{\mu}_{\hat{\gamma}_0}\left(\boldsymbol{d}^{(i)}, \hat{\boldsymbol{\beta}}\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t\right) + \boldsymbol{\mu}_{\hat{\gamma}_1}\left(\boldsymbol{d}^{(i)}, \hat{\boldsymbol{\beta}}\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t\right) + \mu_\delta\left(\boldsymbol{d}^{(i)}, \boldsymbol{\beta}, \boldsymbol{t}\right) \quad (6.8)$$

$$\boldsymbol{\Sigma}_i(j,k) = \sigma^2_{\hat{\gamma}_0}\left(\boldsymbol{d}^{(i)}, \hat{\beta}\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t_j\right)\boldsymbol{U}_0\left(\boldsymbol{d}^{(i)}, t_k\right) + \sigma^2_{\hat{\gamma}_1}\left(\boldsymbol{d}^{(i)}, \hat{\boldsymbol{\beta}}\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t_j\right)\boldsymbol{U}_1\left(\boldsymbol{d}^{(i)}, t_k\right), \forall j,k = 0,1,\cdots,69 \quad (6.9)$$

Given the observation $\{T_{i,l}\}$, where $i = 0,1,\cdots,399$, and $l = 0,1,\cdots,69$, the thermal conductivity coefficients at each spatial location are updated via trained GMMs.

Fig. 6.13 (a) shows an overview of all the 100 calibrated parameters, where mean values are plotted. Besides this, it will be useful to check the correlations among the calibrated parameters. For example, the correlations between parameter k_{50} and the other parameters in the same row $k_{51},\cdots,k_{59}$ (Fig. 6.1) are calculated and plotted in Fig. 6.12. To compare with the calibration result by GP + MCMC from Chapter 5, the calibration is re-plotted in Fig. 6.13 (b). Similar pattern can be found in Fig. 6.13 (a) and (b).

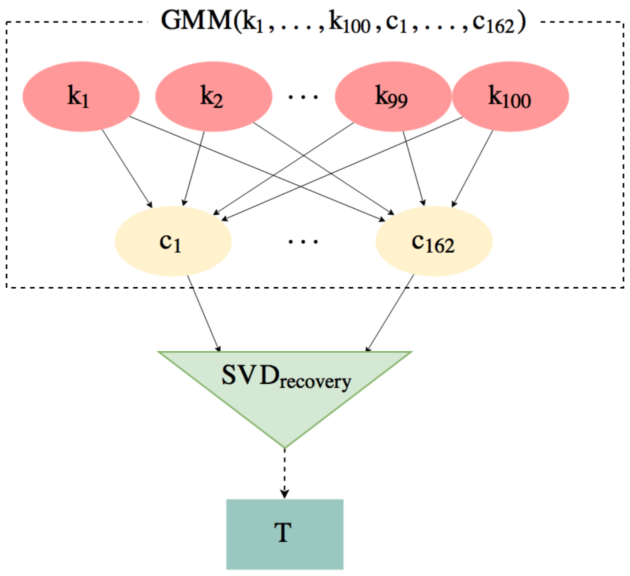


Figure 6.11 Bayesian network for calibration

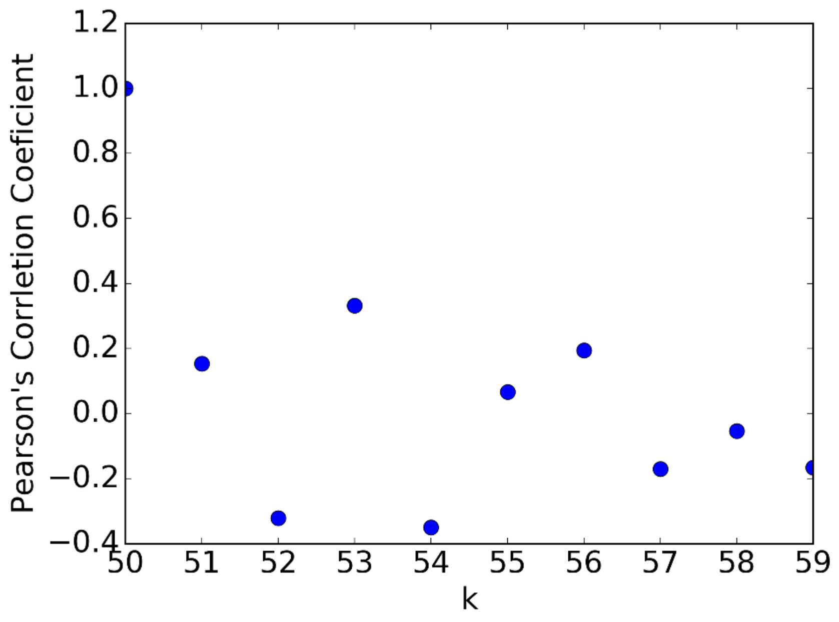
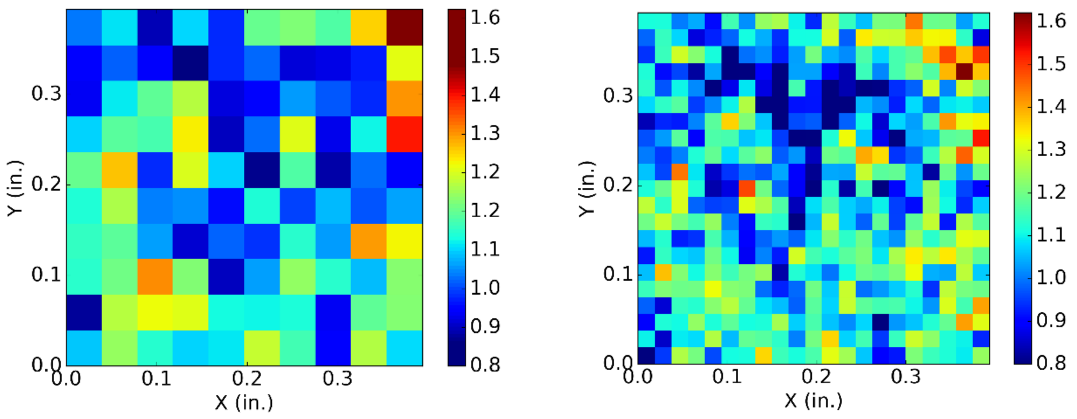


Figure 6.12 Correlation of k_{50} with the other nodes in the same row



(a) (b)

Figure 6.13 Calibration result (mean) shown over the slab top surface (a): by GMM; (b): by GP + MCMC

6.8.6 MapReduce Performance

In this study, 50 nodes were used for parallelization. For the purpose of comparison, computation using the traditional sequential method at a single node was also performed. The configurations of computers are shown in Table 6.1. The comparison between the time cost of the traditional method and MapReduce method is shown in Table 6.2.

Table 6.1 Nodes comparison

Method	CPU (GHZ)	Memory (GB)
Desktop	2.8×2	4
Cluster node	2.3	5

Table 6.2 Time cost of traditional method and MapReduce method

Method	Time (hr.)
Desktop	25.2
Cluster node (scheme 1)	1.64
Cluster node (scheme 2)	10.27
Cluster node (scheme 3)	5.69

MapReduce showed significant computational efficiency (almost 15 times faster). It can be expected that as the number of nodes increases, the time cost could reduce further, but may not be in a linear trend. The reason is that the communication between the master node and slave node also consumes time. The individual time cost shows that surrogate model training points preparation consumes most of the computational resources (Table 6.3).

Compared to the performance of using methodology developed in Chapter 5 (GP surrogate model + MCMC), although the training points preparation spent more time ($36.67\ hrs$ vs. $6\ hrs$), the calibration process greatly save time due to analytical solution is available here (almost no cost vs. $363\ hrs$).

Table 6.3 Time cost of individual steps on desktop

Method	Time (hr.)
FEA model (5 nodes)	36.67
Surrogate model training (50 nodes)	1.64

6.9 Summary

This chapter investigated the MapReduce technique to parallelize the distribution surrogate model, which can be used in model calibration considering a high-dimensional parameter space and in the presence of big data. Three schemes of parallelization were proposed and compared with traditional method (running on a local desktop). It shows that the efficiency is greatly increased due to parallelization. As being a distribution surrogate, GMMs can save a great amount of time in model calibration, since analytical solution can be obtained, compared to response surrogate such as Gaussian process surrogate model used in Chapter 5. It can also be expected after being trained, distribution surrogate model can also outperforms in tasks such as diagnosis and prognosis.

CHAPTER 7

CONCLUSION

This chapter provides the summary of contributions in this study, followed by a discussion of future research needs.

7.1 Summary of Contributions

This dissertation proposed methods to implement big data analytics in structural health monitoring. Four accomplishments are achieved: (1) big data analytics in data processing; (2) big data analytics in structural diagnosis and prognosis uncertainty quantification; (3) big data analytics in high-dimension model parameter calibration; and (4) big data analytics in distribution surrogate model training.

First, a methodology was developed to handle the various steps of data processing in structural health monitoring. MapReduce implementation was proposed to process sensor data of high volume, high velocity, and high variety. Data processing tasks were wrapped in 'mappers' to allow the nodes in cluster to works on the partitions of data set. As an example, image processing for the purpose of structural damage detection was parallelized. However, the developed methodology is applicable for any type of high-volume data in structural health monitoring.

Then, techniques to parallelize structural diagnosis and prognosis with uncertainty quantification were developed. Both forward and inverse problems in uncertainty quantification were investigated with this efficient computational approach. Bayesian methods for the inverse problem of diagnosis, and numerical integration techniques such as Markov chain Monte Carlo (MCMC) simulation and Particle Filter (PF) were parallelized via MapReduce. For the forward problem of prognosis, Monte Carlo sampling on FEA modeling is used to propagate the uncertainties (both aleatory and epistemic) to the future state. Repeated runs of FEA under Monte Carlo sampling were parallelized use MapReduce, thus greatly saving the computational cost.

The system model needs to be updated with latest data in order to perform accurate prognosis of future state. However, the updating is computationally demanding when a model to be calibrated is heterogeneous in its structure or material. A large number of model parameters and large volume of observation data make the computation unaffordable for both surrogate model training and Bayesian calibration. These challenges were addressed through three types of parallelization using the MapReduce technique. The first type of parallelization was to efficiently collect simulation data at the training points for surrogate modeling. Next, the Gaussian process surrogate model training was parallelized using MapReduce. In the third step, parallelization of Markov Chain Monte Carlo (MCMC) technique was studied to efficiently perform Bayesian calibration in the presence of high-volume observation data. In addition to the parallelization of surrogate model training and Bayesian calibration, the singular value decomposition (SVD) method is also employed to reduce the computational effort due to the high-volume data. Furthermore, SVD handled the temporal correlation of the output.

The last accomplishment of this dissertation is big data analytics in distribution surrogate model training. Being a distribution surrogate, a Gaussian mixture model is able to give analytical solutions for prediction and inference, which greatly reduces the cost of calibration of a high-dimensional model with large data. Three parallelization schemes were proposed for GMM training in MapReduce, applicable for different situations (large number of samples or large number of components).

7.2 Future Research Needs

Future research needs to address several extensions. First of all, internal parallelization is preferred to be developed, although some of the accomplishments already contain the internal parallelization such as PF and GMM training. There are two reasons for this point. First, commercial software such as Abqus (which implements FEA) has limitations of license usage. Instead of parallelizing the computation externally (file-wise/data-wise), function-wise decomposition and internal parallelization can be helpful. By doing this, for each input, the model running can be accelerated. Second, in the case of sparse observation data, the running cannot be parallelized by partitioning the data. Instead, internal parallelization by decomposing the functions (such as matrix multiplication) can help.

In addition to the scope of this dissertation (big data analytics in data processing, uncertainty quantification in structural diagnosis and prognosis, high-dimensional model parameters calibration and distribution surrogate model training), there are some related topics which are important in structural health monitoring, which are also time consuming. For example, with respect to the prognosis model, model verification checks how close the model output is to the true solution of the mathematical equation (Szabó and Babuška, 2011).

It is desirable to perform verification before calibration and validation so that the solution approximation errors are accounted for during calibration and validation. Big data analytics techniques in model verification and validation could be investigated in the future.

Diagnosis and model updating are based on the comparison of model prediction against observed data from experiments. Due to limited resources, it is desirable to design the experiments in a way that most information can be obtained from a few experiments (Winer et. al. 1971; Chaloner and Verdinelli, 1995). Bayesian experimental design is one popular method. However, normally this is very computationally expensive, since it needs a double loop of iterative calculations. Furthermore, under limited resources, the performance of a Bayesian update depends significantly on the location of data acquisition. Big data analytics implementation of Bayesian experimental design is another potential research topic to pursue in the future.

REFERENCES

1. Akaike, H. (1974). A new look at the statistical model identification. IEEE Transactions on Automatic Control, 19(6), pp. 716-723.

2. Anastasopoulos, A., Lekou, D. J., & Mouzakis, F. (2012, September). Health monitoring of a neg-micon NM48/750 wind turbine blades with acoustic emission. Proceedings, European Conference on Acoustic Emission Testing & 7th International Conference on Acoustic Emission, University of Granada. Granada, Spain, pp. 12-15.

3. Arulampalam, M. S., Maskell, S., Gordon, N., & Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Transactions on Signal Processing, 50(2), pp. 174-188.

4. Araujo, A., Garca-Palacios, J., Blesa, J., Tirado, F., Romero, E., Samartn, A., & Nieto-Taladriz, O. (2012). Wireless measurement system for structural health monitoring with high time-synchronization accuracy. IEEE Transactions on instrumentation and measurement, 61(3), pp. 801-810.

5. Bagavathiappan, S., Lahiri, B. B., Saravanan, T., Philip, J., & Jayakumar, T. (2013). Infrared thermography for condition monitoring: A review. Infrared Physics & Technology, 60, pp. 35-55.

6. Bao, Y., Beck, J. L., & Li, H. (2010). Compressive sampling for accelerometer signals in structural health monitoring. Structural Health Monitoring, 10(3), pp. 235-

246.

7. Baxes, G. A. (Ed.). (1994). Digital image processing: principles and applications. John Wiley & Sons, Hoboken, New Jersey

8. Bichon, B. J., Eldred, M. S., Swiler, L. P., Mahadevan, S., & McFarland, J. M. (2008). Efficient global reliability analysis for nonlinear implicit performance functions. AIAA J, 46(10), pp. 2459-2468.

9. Bishop, C. M. (2006). Pattern recognition and machine learning. Springer, New York, NY.

10. Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992, July). A training algorithm for optimal margin classifiers. Proceedings, Fifth Annual Workshop on Computational Learning Theory, ACM, pp. pp. 144-152.

11. Cai, G., & Mahadevan, S. (2016). Big data analytics in structural health monitoring. International Journal of Prognostics and Health Management, 7, 2016.

12. Chakraborty, D., Kovvali, N., Wei, J., PapandreouSuppappola, A., Cochran, D., & Chattopadhyay, A. (2009). Damage classification structural health monitoring in bolted structures using time-frequency techniques. Journal of Intelligent Material Systems and Structures, 20(11), pp. 289-305.

13. Chaloner, K., & Verdinelli, I. (1995). Bayesian experimental design: A review. Statistical Science, pp. 273-304.

14. Chatzi, E. N., & Smyth, A. W. (2013). Particle filter scheme with mutation for the estimation of time-invariant parameters in structural health monitoring applications.

Structural Control and Health Monitoring, 20(7), 1081-1095.

15. Chen, W. Y., Song, Y., Bai, H., & Lin, E. Y., C. J.and Chang. (2011). Parallel spectral clustering in distributed systems. IEEE transactions on pattern analysis and machine intelligence, 33(3), pp. 568-586.

16. Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. Communications of the ACM, 5(1), pp. 107-113.

17. Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B (methodological), pp. 1-38.

18. Desjardins, S. L., Londono, N. A., Lau, D. T., & Khoo, H. (2006). Real-time data processing, analysis and visualization for structural monitoring of the confederation bridge. Advances in Structural Engineering, 9(1), pp. 141-157.

19. Di Ianni, T., De Marchi, L., Perelli, A., & Marzani, A. (2015). Compressive sensing of full wave field data for structural health monitoring applications. IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency control, 62(7), 1373-1383.

20. Doucet, A., De Freitas, N., Murphy, K., & Russell, S. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., pp. 176-183.

21. Farrah, S., Ziyati, H. E. M. E. H., & Ouzzif, M. (2015). An approach to analyze large

scale wireless sensors network data. Measurements, 2(5), pp. 7-12.

22. Farrar, C. R., Doebling, W., S., & Nix, D. A. (2001). Vibration-based structural damage identification. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 359(1778), pp. 131-149.

23. Farrar, C. R., & Worden, K. (2007). An introduction to structural health monitoring. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 365(1851), pp. 303-315.

24. Feldman, D., Faulkner, M., & Krause, A. (2011). Scalable training of mixture models via coresets. In Advances in Neural Information Processing Systems, pp. 2142-2150.

25. Gandhi, T., Chang, R., & Trivedi, M. M. (2007). Video and seismic sensor-based structural health monitoring: Framework, algorithms, and implementation. IEEE Transactions on Intelligent Transportation Systems, 8(2), pp. 169-180.

26. Ghanem, R., & Spanos, P. D. (1990). Polynomial chaos in stochastic finite elements. Journal of Applied Mechanics, 57(1), pp. 197-202.

27. Gilks,W.R.(2005). Markovchainmontecarlo. EncyclopediaofBiostatistics.

28. Haldar, A. & Mahadevan, S. (2000). Probability, reliability, and statistical methods in engineering design, Vol. 1. Wiley New York.

29. Heckerman, D. (1998). A tutorial on learning with Bayesian networks. Nato Asi Series D Behavioural And Social Sciences, 89, pp. 301-354.

30. Humphrey, M., Beekwilder,N., Goodall,J.L., & Ercan, M.B. (2012). Calibration of

water shed models using cloud computing. E-science (eScience), 2012 IEEE 8th International Conference On, IEEE, pp. 1-8.

31. Huang, S., Mahadevan, S., & Rebba, R. (2007). Collocation-based stochastic finite element analysis for random field problems. Probabilistic engineering mechanics, 22(2), 194-205.

32. Jain, R., Kasturi, R., & Schunck, B. G. (1995). Machine vision. McGraw-Hill, New York, 5.

33. Jensen, F. V. (1996). An introduction to Bayesian networks. UCL press, London, 210, pp. 1-178

34. Jin, H., Wong, M. L., & Leung, K. S. (2005). Scalable model-based clustering for large databases based on data summarization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(11), pp. 1710-1719.

35. Kallinikidou, E., Yun, H. B., Masri, S. F., Caffrey, J. P., & Sheng, L. H. (2013). Application of orthogonal decomposition approaches to long-term monitoring of infrastructure systems. Journal of Engineering Mechanics, 139(6), pp. 678-690.

36. Karabinis,A. & Rousakis, T. (2002). Concrete confined by frp material: a plasticity approach. Engineering Structures, 24(7), pp. 923-932.

37. Kezia, S. P. & Mary, A. V. A. (2016). "Prediction of rapid floods from big data using mapreduce technique." Global Journal of Pure and Applied Mathematics, 12(1), pp. 369-373.

38. Kiepert, J., & Loo, S. M. (2012). A unified wireless sensor network framework. In

Systems conference (syscon), IEEE International, pp. 1-6.

39. Kumar, N. P., Satoor, S., & Buck, I. (2009, June). Fast parallel expectation maximization for Gaussian mixture models on GPUs using CUDA. In High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference, pp. 103-109.

40. Kwedlo, W. (2014, February). A parallel EM algorithm for Gaussian mixture models implemented on a NUMA system using OpenMP. In Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd IEEE Euromicro International Conference, pp. 292-298

41. Landau, W. & Niemi, J. (2016). A fully Bayesian strategy for high-dimensional hierarchical modeling using massively parallel computing. arXiv preprint arXiv:1606.06659.

42. Lefèvre, S., Volz, S., Saulnier, J.-B., Fuentes, C., & Trannoy, N. (2003). Thermal conductivity calibration for hot wire based dc scanning thermal microscopy. Review of Scientific Instruments, 74(4), pp. 2418-2423.

43. Liang, C. (2016). Multidisciplinary Analysis and Optimization under Uncertainty, Doctoral dissertation, Vanderbilt University.

44. Liang, C., & Mahadevan, S. (2016). Stochastic multidisciplinary analysis with high-dimensional coupling. AIAA Journal.

45. Ling, Y., Mullins, J., & Mahadevan, S. (2014). Selection of model discrepancy priors in Bayesian calibration. Journal of Computational Physics, 276, pp. 665-680.

46. Lopez-Higuera, J. M., Cobo, L. R., Incera, A. Q., & Cobo, A. (2011). Fiber optic sensors in structural health monitoring. Journal of Lightwave Technology, 29(4), pp. 587-608.

47. McLachlan, G., & Peel, D. (2000). Mixtures of factor analyzers. Finite Mixture Models, pp. 238-256.

48. Madsen, H. (2003). Parameter estimation in distributed hydrological catchment modelling using automatic calibration with multiple objectives. Advances in Water Resources, 26(2), pp. 205-216.

49. Mahadevan, S., Adams, D., & Kosson, D. (2014). Challenges in concrete structures health monitoring. In In proceedings, annual conference of the prognostics and health management society.

50. McLachlan, G. J., & Krishnan, T. (1997). Wiley series in probability and statistics. The EM Algorithm and Extensions, Second Edition, pp. 361-369.

51. Mcnicholas, P. D., & Murphy, T. B. (2008). Parsimonious Gaussian mixture models. Statistics and Computing, 18(3), pp. 285-296.

52. Meeker, W. Q. & Hong, Y. (2014). Reliability meets big data: opportunities and challenges. Quality Engineering, 26(1), pp. 102-116.

53. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. The Journal of Chemical Physics, 21(6), pp. 1087-1092.

54. Nagy, P. B. (2016). Electromagnetic nondestructive evaluation. Ultrasonic and

Electromagnetic NDE for Structure and Material Characterization: Engineering and Biomedical Applications, 169.

55. Nair, A., & Cai, C. S. (2010). Acoustic emission monitoring of bridges: Review and case studies. Engineering Structures, 32(6), pp. 1704-1714.

56. Nannapaneni, S. & Mahadevan, S. (2016). Reliability analysis under epistemic uncertainty. Reliability Engineering & System Safety, 155, pp. 9-20.

57. Nath, P., Hu, Z., & Mahadevan, S. (2017). “Bayesian calibration of spatially varying model parameters with high-dimensional response.” 19th AIAA Non-Deterministic Approaches Conference, 1775.

58. Neal, R. M. (2003). Slice sampling. Annals of statistics, pp. 705-741.

59. Neiswanger, W., Wang, C., & Xing, E. (2013). Asymptotically exact, embarrassingly parallel MCMC. arXiv preprint arXiv:1311.4780.

60. Nelsen, R. B., An introduction to copulas, Springer, New York, 1999.

61. Naus, D. J. (2009). The management of aging in nuclear power plant concrete structures. Journal of Metals, 61(7), pp. 35-41.

62. Orlande, H., Colaço, M., Dulikravich, G., Vianna, F., da Silva, W., da Fonseca, H., & Fudym, O. (2011). Tutorial 10 kalman and particle filters. Advanced Spring School: Thermal Measurements & Inverse Techniques 5 (Mesures en Thermiques et Techniques Inverses, Roscoff, FR), pp. 1-39.

63. Papasalouros, D., Tsopelas, N., Ladis, I., Kourousis, D., Anastasopoulos, A., Lekou,

D., & Mouzakis, F. (2012). Health monitoring of a neg-micon nm48/750 wind turbine blade with acoustic emission. Proceedings of the 30th European Conference on Acoustic Emission (EWGAE) & 7th International Conference on Acoustic Emission, Granada, Spain, pp. 12-15.

64. Park, S., Ahmad, S., Yun, C. B., & Roh, Y. (2006). Multiple crack detection of concrete structures using impedance-based structural health monitoring techniques. Experimental Mechanics, 46(5), pp. 609-618.

65. Rajashekhar, M. R., & Ellingwood, B. R. (1993). A new look at the response surface approach for reliability analysis. Structural Safety, 12(3), pp. 205-220.

66. Rasmussen, C. E., & Williams, C. K. (2006). Gaussian processes for machine learning. MIT press, Cambridge, Massachusetts.

67. Reynolds, D. (2015). Gaussian mixture models. Encyclopedia of biometrics, pp. 827-832.

68. Roberts, G. O. & Rosenthal, J. S. (2006). Harris recurrence of metropolis-within-Gibbs and trans-dimensional Markov chains. The Annals of Applied Probability, 16(4), pp. 2123-2139.

69. Rose, C., & Smith, M. D. (1996). The multivariate normal distribution. Mathematica Journal, 6(1).

70. Roshandeh, A. M., Poormirzaee, R., & Ansari, F. S. (2014). Systematic data management for real-time bridge health monitoring using layered big data and cloud computing. International Journal of Innovation and Scientific Research, 2(1), pp. 29-

39.

71. Roux, E., & Bouchard, P. O. (2015). On the interest of using full field measurements in ductile damage model calibration. International Journal of Solids and Structures, 72, pp. 50-62.

72. Roux, S., Réthoré, J., & Hild, F. (2009). Digital image correlation and fracture: an advanced technique for estimating stress intensity factors of 2D and 3D cracks. Journal of Physics D: Applied Physics, 42(21), 214004.

73. Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., & Tarantola, S. (2008). Global sensitivity analysis: the primer. John Wiley & Sons.

74. Sankararaman, S. & Mahadevan, S. (2015). Integration of model verification, validation, and calibration for uncertainty quantification in engineering systems. Reliability Engineering & System Safety, 138, pp. 194-209.

75. Santner, T. J., Williams, B. J., & Notz, W. I. (2013). The design and analysis of computer experiments. Springer Science & Business Media.

76. Saouma, V. & Perotti, L. (2006). Constitutive model for alkali-aggregate reactions. ACI Materials Journal, 103(3), pp. 194.

77. Schwarz, G. (1978). Estimating the dimension of a model. The Annals of Statistics, 6(2), pp. 461-464.

78. Sohn, H., Farrar, C., Hunter, N., & Worden, K. (2001, Jan.). Applying the lanl statistical pattern recognition paradigm for structural health monitoring to data from

a surface-effect fast patrol boat (Tech. Rep.).

79. Szabó, B., & Babuška, I. (2011). Introduction to finite element analysis: formulation, verification and validation (Vol. 35). John Wiley & Sons.

80. Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. Journal of Machine Learning Research, 1(Jun), pp. 211-244.

81. Tran, C. (1868). “Structural-damage detection with big data using parallel computing based on mpsoc.” International Journal of Machine Learning and Cybernetics, pp. 1-11.

82. Ulm, F.-J., Coussy, O., Kefei, L., & Larive, C. (2000). Thermo-chemo-mechanics of asr expansion in concrete structures. Journal of Engineering Mechanics, 126(3), pp. 233-242.

83. Winer, B. J., Brown, D. R., & Michels, K. M. (1971). Statistical principles in experimental design (Vol. 2). McGraw-Hill, New York.

84. Xu, P. (1998). Truncated svd methods for discrete linear ill-posed problems. Geophysical Journal International, 135(2), pp. 505-514.

85. Yan, F., Royer, R. L., & Rose, J. L. (2010). Ultrasonic guided wave imaging techniques in structural health monitoring. Journal of Intelligent Material Systems and Structures, 21(3), pp. 377-384.

86. Yu, L. (2012). Acoustic Emission Source Localization on Concrete Structures with Focusing Array Imaging. In 6th European Workshop on Structural Health Monitoring.

87. Yu, L. & Lin, J.-C. (2015). Cloud computing-based time series analysis for structural damage detection. Journal of Engineering Mechanics, C4015002.

88. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2-2.

89. Zhang, J., Qiu, H., Shamsabadi, S. S., Birken, R., & Schirner, G. (2014, Jul.). Sirom3–a scalable intelligent roaming multi-modal multi-sensor framework. In 38th IEEE International Conference on Computers, Software and Applications, pp. 446-455.

90. Zhong, L., Tang, K., Li, L., Yang, G., & Ye, J. (2014). An improved clustering algorithm of tunnel monitoring data for cloud computing. The Scientific World Journal, 2014.