MAXIMIZING SERVICE UPTIME OF SMARTPHONE-BASED DISTRIBUTED REAL-TIME AND EMBEDDED SYSTEMS

By

Anushi Shah

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

December, 2010

Nashville, Tennessee

Approved:

Dr. Aniruddha Gokhale

Dr. Abhishek Dubey

To my husband Vaibhav and my parents
for giving me immense love and support over the years

ACKNOWLEDGMENTS

I would like to acknowledge the following people who have helped me in one or the other way during my graduate studies at Vanderbilt University. First of all, my advisor Dr. Aniruddha Gokhale for giving me an opportunity to work in the DOC group, encouragement, support and sharing his knowledge during my thesis work.

Dr. Jules White whose earlier research work inspired to formulate the ideas for my thesis. Dr. Abhishek Dubey for providing valuable comments and feed backs in reviewing this thesis. Brian Dougherty, Tripti Saxena, Kyoungho An, Nilabja Roy and all DOC group members for discussing and sharing their knowledge.

On a personal note, my parents, my in-laws, my brother, my sister and my sister-in-law for being so encouraging and supportive of me pursuing graduate studies.

Last but not least, my husband Vaibhav for being so understanding and being my pillar of strength.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The unprecedented growth in smartphone technology is giving rise to new applications that illustrate non-conventional usage of smartphones [1]. For example, these applications may include situational awareness in military-centric operations (*e.g.*, the DARPA Transformative Apps program), emergency services, disaster search-and-recovery, and intelligent transportation. Consider, for example, natural disasters of 2010 like the Haiti earthquake or the massive flooding in the state of Tennessee. In both these situations, most of the infrastructure, such as the roads and phone services (both landline and cellular), and utilities, such as gas and electricity, were rendered unavailable. A number of instances of smartphone usage for survival have come to light in the days following the calamity. It is conceivable, therefore, to think of forming *ad hoc* networks of smartphones carried by search-and-rescue teams as the best means in these circumstances to identify survivors trapped under the debris or those trapped in their houses due to raging flood waters, and coordinate the rescue operations.

To operationalize smartphone-based search-and-rescue missions, it is necessary for the collection of smartphones involved in the mission to be able to support a group of real-time services that provide distributed sensing operations, data correlation capabilities stemming from acquisition of distributed streams of images, audio and video, and location-based services. However, since these smartphones have limited battery life and hardware resources, keeping the collective set of services that make up the mission capabilities up and running for the maximum amount of time is crucial for maximizing the chances of finding more survivors. Maximizing the mission lifespan is important because the smartphones operated by first responders are often deployed in environments where readily replenishing the resources, such as batteries, is infeasible. Despite these constraints, key quality of service (QoS) requirements of real-time and reliable dissemination of information to the concerned stakeholders, such as first responders in search-and-rescue missions, must be met.

The requirements outlined above can be met by effectively deploying the services that make up the mission on the collection of smartphones involved in the mission. Here ad hoc network can be formed by smartphones which is self-configuring and self-organizing as no physical infrastructure is available for forming centrally administered wireless network. Hence we are assuming that the appropriate ad hoc routing protocols like AODV [12], DYMO [13], etc are available for routing of data to/from ad hoc network. Also, each node in the ad hoc network has equal probability of acting as hosts as well as routers to route data to/from other nodes in the network. The reason is that considerable amount of battery power is consumed in routing data to/from network to/from the outside network. Thus if a particular node has a higher probability of acting a router, then its battery power will be drained out faster, which can render the entire distributed application unoperational earlier than its maximized service uptime. However, such a deployment problem is hard for two reasons. First, assuring the timely and reliable dissemination of information in operating environments where availability of resources, such as networks, is unpredictable requires deploying the individual services on the collection of smartphones in a way that will ensure the schedulability of the services while efficiently using the scarce resources. Secondly, the rate of drain of smartphone battery charge adds a new dimension of challenges to an already challenging problem because battery drain is often dictated by the amount of computation and communication activities.

In this thesis we focus on solving the *service uptime maximization* problem, which is the problem of ensuring that the operational capability of the mission provided by the collection of services deployed on the group of smartphones remains up and running for the maximum duration of time. In other words, it is necessary to minimize the rate at which the smartphone batteries drain themselves. Since every service (and its software components) of the mission consumes different computational and communication resources of the smartphone, battery drain is impacted differently. Hence, the service uptime maximization problem requires solving the deployment problem that minimizes battery drain (or preserves the battery charge) while also satisfying the QoS requirements.

To address these challenges, we present a deployment framework called *SmartDeploy,* which extends the earlier work on ScatterD [8] done in the group. ScatterD combined bin-packing heuristics with evolutionary algorithms to minimize power consumption in nodes. It overcame the limitations of applying each of these algorithms in isolation. In particular, ScatterD provided a first-fit heuristic bin packer which places each item into the first available bin in which it will fit. In the case of maximizing service uptime, the software components of the services must be deployed in a way that minimizes battery drain on each smartphone. A first-fit heuristic may not necessarily find the right solution to our problem. Consequently, SmartDeploy provides a framework that can be strategized with the desired bin packing heuristic along with a strategizable framework to plug in the desired evolutionary algorithm so that a variant of the hybrid algorithm can be synthesized.

To solve the service uptime maximization problem, SmartDeploy is strategized with the worst-fit bin packer which ensures that services are load balanced across the collection of smartphones used in the mission in a way that minimizes battery drain while also delivering the QoS. The evolutionary algorithm generates initial random vectors and evaluates them using a fitness function. In this thesis we limit ourselves to off line deployment of services assuming that the rescue missions and their parameters are planned *a priori*. The case of determining an effective deployment at runtime is orthogonal to the focus of this thesis and is the focus of future work, which will require additional runtime protocols involving message exchanges among participating smartphones. We believe that the polynomial runtime complexity of SmartDeploy can make it a promising approach even at runtime.

CHAPTER II

RELATED WORK

Heuristic techniques are commonly used for large size optimization problems. They are used to generate good solutions without exhaustive search which is time consuming. These techniques work iteratively where each step depends on the previous step and hence are called heuristic. Hill Climbing algorithms use local information about a search space to find optima. However, for large problem sizes and complicated functions, they tend to get stuck in local optima. Simulated annealing is a heuristic technique for global optimization where it starts with a random state (solution) and iteratively moves towards better solution. However, in each iteration it compares one solution with the previous ones. Evolutionary algorithms which are also heuristic algorithms use pool of solution in each iteration for comparison. Hence there is a better chance of achieving a good result.

Xiaoling et al are amongst the first to use evolutionary algorithm for deployment optimization problem [15] in ad-hoc sensor network. They optimized the coverage in sensor network. They compared particle swarm optimization (PSO) with the genetic algorithm in terms of faster convergence rate. However, they did not address performance of the evolutionary algorithms used when design space and constraints increases. Our goal is maximizing service uptime of distributed applications comprising a large design space of hundreds of nodes and hundreds of software components. Moreover, the design space in our case is tightly constrained based on hardware and software resources availability.

François et al developed Choco [11], a Java library for constraint satisfaction problems (CSP) and constraint programming (CP). It is built on an event-based propagation mechanism with backtrackable structures. Since it is based on CSP approach, it leads to exhaustive search in the worst case. Hence, it is not scalable with problem sizes handled by SmartDeploy.

Howard et al developed an algorithm [6] for deploying members of a robot team into an unknown environment. However they assumed unavailability of GPS sensors on robots and required maintaining line-of-sight contact amongst the team members. In our case we consider a network of smartphones, which involves availability of GPS and other sensors. Thus we do not need to maintain line-of-sight with the other devices.

Howard et. al. also developed an incremental and greedy algorithm [7] for mobile sensor network. Their test results, however, assumed the nodes to be homogeneous and the scalability of the algorithm was tested up to 50 nodes. In our case we consider heterogeneous devices in terms of power capacity, memory, CPU, etc. Moreover, we consider deployment of hundreds of software components on hundreds of devices.

Dougherty et. al. developed a deployment algorithm called BLITZ [4] that minimizes the computing infrastructure required to host real-time systems. The algorithm uses first-fit heuristics of bin packing algorithm that minimizes number of processors. However, the service uptime maximization requires the use of worst-fit bin packing heuristics.

White et. al. developed a spatial deployment algorithm called ScatterD [8] that minimizes power consumption in real-time systems. It is a hybrid algorithm that combines first-fit bin packing heuristics with evolutionary algorithms (genetic and particle swarm optimization algorithms). The first-fit bin packing algorithm places the items on to the first available bin till it gets exhausted and then selects the next bin. This heuristic does not place item on the emptiest existing bin which is more suitable for maximizing service uptime. SmartDeploy extends ScatterD to provide a strategizable framework and applies worst-fit bin packing for the Service Uptime Maximization problem.

CHAPTER III

MOTIVATING EXAMPLE

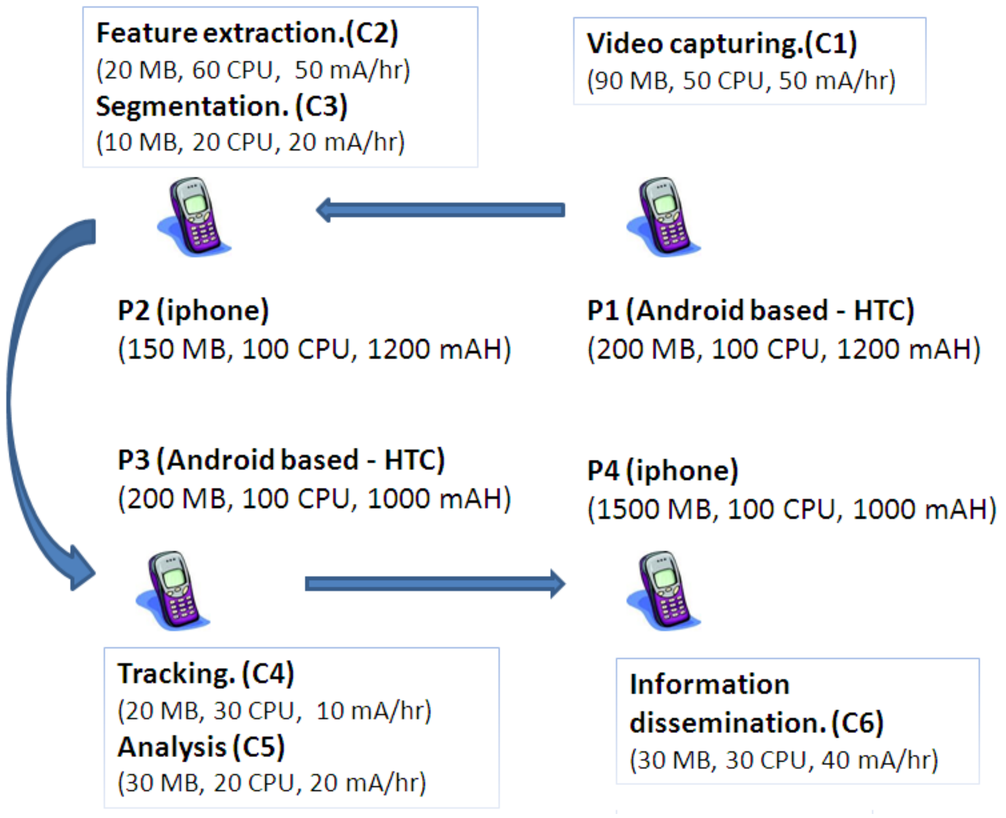
Feature extraction.(C2)
(20 MB, 60 CPU, 50 mA/hr)
Segmentation. (C3)
(10 MB, 20 CPU, 20 mA/hr)
Video capturing.(C1)
(90 MB, 50 CPU, 50 mA/hr)
P2 (iphone)
(150 MB, 100 CPU, 1200 mAH)
P1 (Android based - HTC)
(200 MB, 100 CPU, 1200 mAH)
P3 (Android based - HTC)
(200 MB, 100 CPU, 1000 mAH)
P4 (iphone)
(1500 MB, 100 CPU, 1000 mAH)
Tracking. (C4)
(20 MB, 30 CPU, 10 mA/hr)
Analysis (C5)
(30 MB, 20 CPU, 20 mA/hr)
Information
dissemination. (C6)
(30 MB, 30 CPU, 40 mA/hr)

Figure III.1: A Distributed Video Recognition Service for Disaster Monitoring

In this section we use an example of a video recognition service for disaster monitoring as a case study to highlight the challenges in maximizing the service uptime for smartphone-based distributed, real-time systems. Figure III.1 shows an example of a distributed video recognition service used in disaster monitoring and recovery. The service comprises of different software components like video capturing (C1), segmentation (C2), feature extraction (C3), tracking (C4), activity analysis (C5) and information dissemination (C6). Each of these software component has different hardware resource requirements, such as memory and CPU, and different power consumption rate. For simplicity, we have shown one such distributed service (video recognition) consisting of six software components and four smartphones for disaster monitoring. Out of four smartphones used, two of them are Android-based HTC phones and the other two are iPhones. Software components C1, C4, and C5 can be executed only on Android-based smartphones, while software components C2, C3, and C6 can be executed only on iPhones.

In general, a disaster monitoring service can be composed of a combination of services such as distributed image recognition and distributed location-based services. Such a comprehensive service can consist of hundreds of software components deployed onto hundreds of smartphones. The deployment plan, which comprises a mapping of the software components of the services to the smartphones, should meet both the hardware resources constraints and power constraints such that the service can last for as much time as possible while also meeting the real-time application requirements.

CHAPTER IV

CHALLENGES IN MAXIMIZING SERVICE UPTIME FOR SMARTPHONES

In this chapter we use our motivating example of the distributed video recognition service (see Chapter III) to highlight the challenges in finding the deployment plan which maximizes service uptime. Although the mobile environment made up of smartphones is attractive to realize distributed disaster management services, multiple systemic issues impede the total lifetime of such services making it hard to design and deploy the services. In this section we delve into understanding these impediments.

Challenge 1: Dealing with Complex hardware/software design constraints In our case study example of the distributed video recognition service, its software components have different hardware and software resource requirements. For example, the video capturing component requires high memory and communicational power as it stores the captured video and sends it to the phone hosting feature extraction and segmentation components. The feature extraction and segmentation components require high CPU and computational power as they run complex algorithms based on extraction and segmentation on the video. The tracking and activity analysis components are involved in significant communication activities that consume battery power as they constantly communicate with the phone hosting information dissemination component. A disaster monitoring system comprises many distributed applications consisting of hundreds of smartphones and hundreds of software components hosted on them. How these software components are deployed on these smartphones will determine how long the overall mission will last because the uptime of the mission depends on how long the batteries will last.

In general, network embedded devices like smartphones have limited battery power and limited hardware resources like CPU and memory. Moreover, the software components deployed on these devices consume power at different rates, which is governed by the computation and communication activities induced by the software components. The amount of time a software component runs is directly proportional to the amount of battery power available to it with sufficient hardware resources. Thus, the power consumption rate of these software components, and what devices they get deployed on are the key factors that affects the service uptime.

Given that a mission is realized by distributing its services across a group of smartphones, keeping the entire distributed application up for a longer duration is challenging because even if one of the smartphone's battery is exhausted, then the software components deployed on it are no longer available which makes the overall distributed system no longer work. Thus, a deployment plan should be generated such that each of the software components gets maximum available power and sufficient hardware resources which will maximize the overall service uptime of the mission. In generating such a deployment plan, we must consider both the computational and communication power consumption rates of the software components. Some components may have higher power consumption rate due to high amount of computations involved, while some components engage in more communication activities that impacts the power consumed. The frequency of interaction between software components affects the amount of bandwidth consumed by them, which in turn affects their power consumption rate.

Challenge 2 : Dealing with heterogeneity of available resources and execution constraints Our case study example illustrates heterogeneity in the smartphone hardware and operating systems. It is conceivable that embedded devices such as smartphones used in mission-critical applications such as disaster search and rescue management have different available hardware resources like CPU type, available memory, and lifetime of battery.

Due to this heterogeneity, certain software can execute on only certain devices. For example, *smartphone apps* developed for *iPhones* cannot execute on Android-based phones. As outlined in Challenge 1, the deployment topology of these mission-critical systems must address various design constraints like power capacity, memory, and CPU, which is a hard problem. The problem becomes even harder with the heterogeneity of the platforms and the software execution constraints. In the case study example, there are two Android-based HTC phones and the other two are iphones. Moreover, independent software components for video capturing, tracking and analysis can execute only on Android-based phones. Similarly, feature extraction, segmentation and information dissemination can execute only on iPhones. Such constraints affect the deployment plan which in turn affects maximizing service uptime.

Challenge 3: Dealing with scale of the system The case study example of distributed video recognition service is comprised of four devices hosting six software components which means there exist 6^4 possible deployment plans. Several optimization techniques are available to solve the deployment challenges explored in Challenges 1 and 2 described above. The solutions can be characterized and solved using constraint satisfaction programming (CSPs) [14], integer programming [3] and Bender's decomposition [5].

Although our case study represents a very small problem size which can be solved by bin-packing heuristics, integer programming or evolutionary algorithms, typical mission critical applications will comprise several hundreds of devices and many more software components. Thus, when the problem size scales to 300^{100} or even more and moreover considering additional hardware and software design constraints, as outlined in Challenges 1 and 2, many of the known techniques cannot readily scale to hundreds of software components and hundreds of devices. In other words, the solutions are computationally very expensive to obtain.

Bin packing heuristics have been developed to overcome these challenges to produce valid deployment plans, however, these plans do not necessarily produce the optimal solutions for large problem sizes. Evolutionary algorithms are commonly used in deployment optimization problems. However their performance degrades when the solution space is huge and has tight constraints that leads a large number of invalid points in the search space. The criticality of the application scenario we are investigating and the fact that we focus on offline solutions to finding the right deployment topologies, it is desirable to achieve a near-optimal solution. Moreover, formulating the objective function and the constraints is yet another challenge system developers will face.

CHAPTER V

BIN-PACKING HEURISTICS FOR SERVICE UPTIME MAXIMIZATION

In this chapter, bin-packing heuristics and its variants are described. The problem of packing a set of items into a number of bins such that the total weight or volume does not exceed some maximum value is called bin-packing problem. Various heuristics of bin-packing algorithms are used for solving bin-packing problem like first-fit, worst-fit, and best-fit.

First-fit bin packing algorithm : The First-fit algorithm places a new object in the first available bin that still has room.

Best-fit bin packing algorithm : The Best-fit algorithm places a new object in the fullest bin that still has room.

Worst-fit bin packing algorithm : The Worst-fit algorithm places a new object in the emptiest existing bin.

For the service uptime maximization problem, we use the worst-fit heuristic of bin packing algorithm. The reason is that in order to maximize the service uptime, the software components should be deployed onto the device on which it can run for maximum amount of time. Thus the worst-fit bin packing algorithm defines the placement of items into the largely empty existing bin. In this way the software components are deployed evenly across the available devices such that they get maximum available power along with sufficient hardware resources. As a result maximized service uptime is achieved. However, as the problem size increases, it tends to give a valid solution but not necessarily an optimal one.

CHAPTER VI

EVOLUTIONARY ALGORITHMS FOR SERVICE UPTIME MAXIMIZATION

Evolutionary algorithms [2], are meta-heuristic optimization algorithms which are generic population based, *i.e.*, they involve a search from a population of solutions, not from a single point. The individuals in the population are candidate solutions of the optimization problem. In each iteration, a fitness function is used to evaluate the candidate solutions and propagates the evolution of the population. Particle swarm optimization(PSO) [9] and genetic algorithm [10] are two such evolutionary algorithms.

1. Particle swarm optimization (PSO) It is a stochastic optimization technique based on population. Here, each particle is a random initial solution in the search space, *i.e.*, random initial topology vector $\vec{vi}$. In each iteration, the particles are evaluated using fitness function (objective function) $F(\vec{vi})$ and the best value of each particle is maintained. Each particle's best value is compared with the global best value. The global best value gives the solution for the fitness function. At the end of each iteration, each particle's position and velocity is updated based on the global best value. This process is repeated till the number of iterations are reached or the process converges to a single solution. Figure VI.1 shows the PSO algorithm.

2. Genetic algorithms It is also a stochastic technique based on population. Here, the initial random solutions are candidate solutions (individuals or creatures) that are encoded by population of strings (chromosomes). In each generation, the individuals are evaluated using fitness function $F(\vec{vi})$, multiple individuals are stochastically selected from current population on the basis of fitness function, reproduced by crossover or mutation which

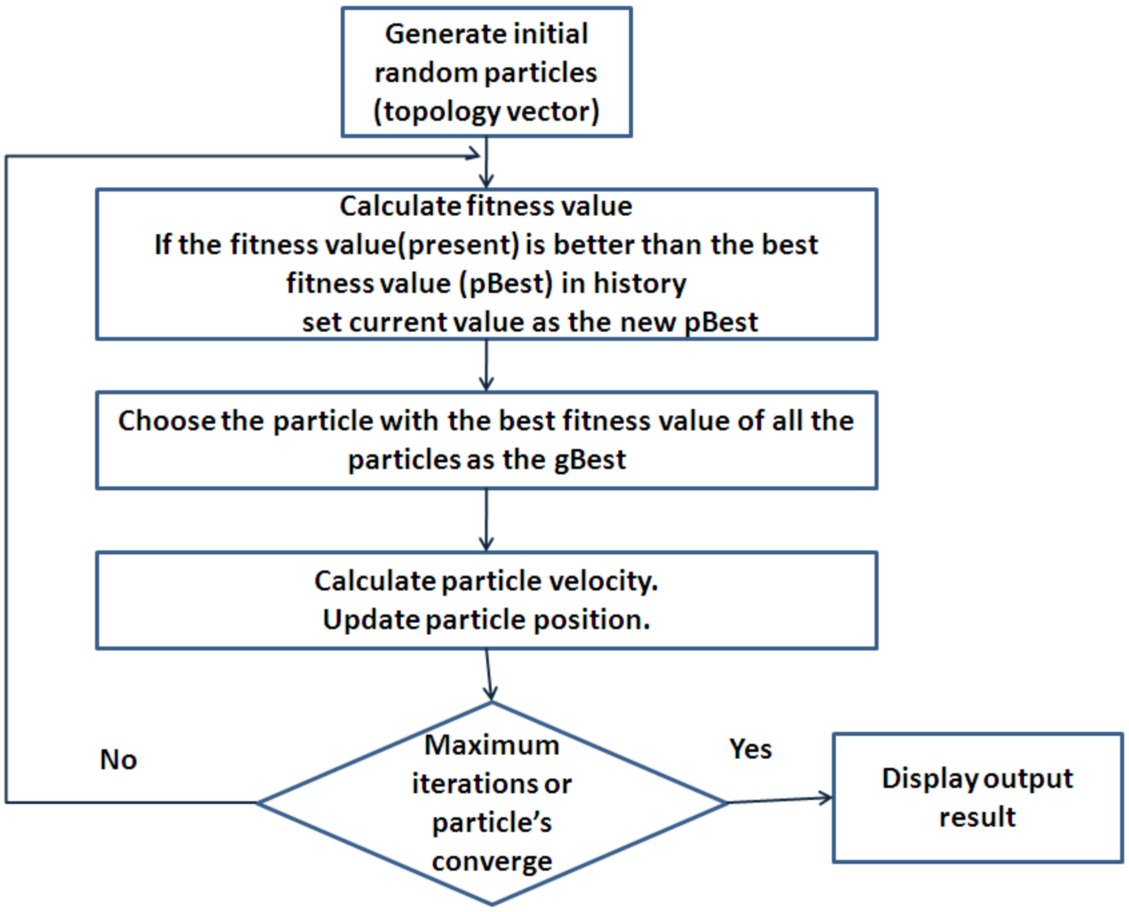
Generate initial random particles (topology vector)
Calculate fitness value
If the fitness value(present) is better than the best fitness value (pBest) in history set current value as the new pBest
Choose the particle with the best fitness value of all the particles as the gBest
Calculate particle velocity.
Update particle position.
Maximum iterations or particle's converge
No
Yes
Display output result

Figure VI.1: Particle swarm optimization (PSO)

forms new population. This new population is evaluated using fitness function and the process continues till the number of iterations are reached or the process converges to a single solution. Figure VI.2 shows the genetic algorithm.

In general, as shown in Figure VI.3 [8], members of solution topologies (particles or genes) are represented as vectors where the vector components denote the position of particles or genes. The spatial deployment topology as shown in the figure is represented as $\vec{V} = [1,2,2]$ which in turn represents the index positions of the software components deployed onto the hardware node, *i.e.*, software components 1, 2, and 3 are deployed onto device 1, 2 and 2, respectively. As the evolution proceeds, the deployment topology vector

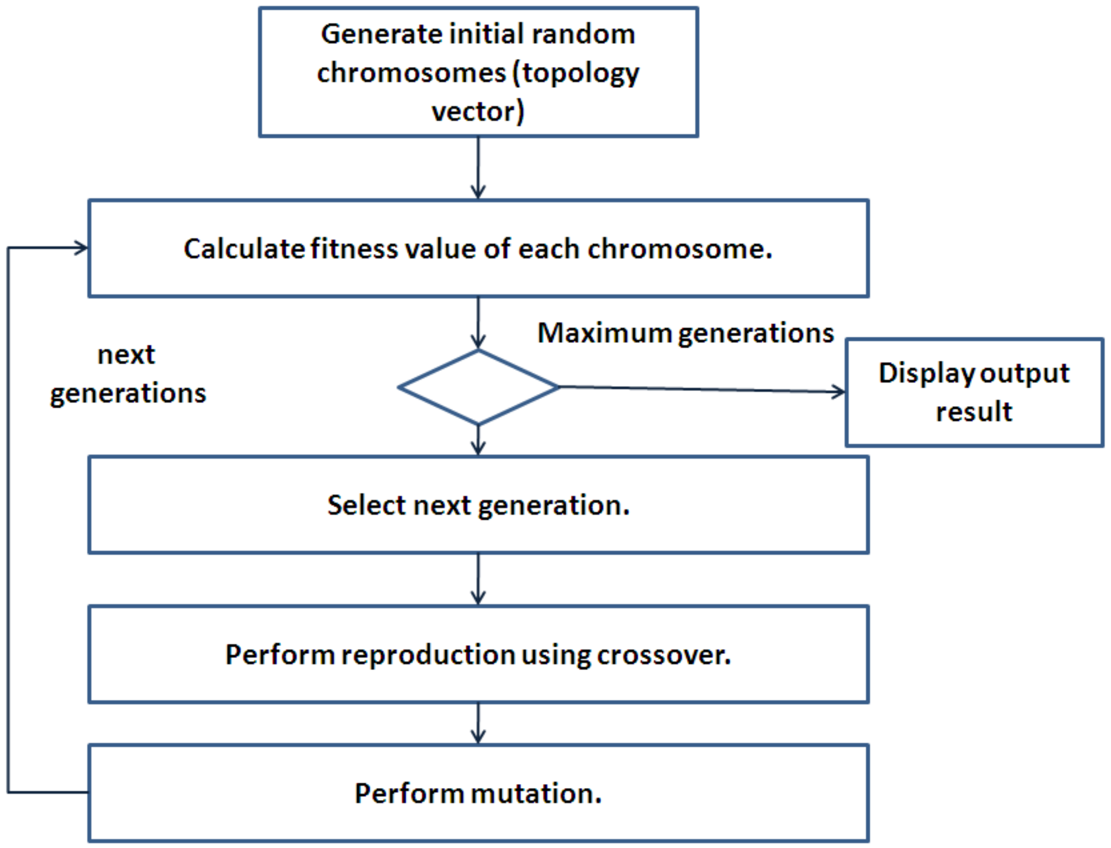
Generate initial random chromosomes (topology vector)
Calculate fitness value of each chromosome.
next generations
Maximum generations
Display output result
Select next generation.
Perform reproduction using crossover.
Perform mutation.

Figure VI.2: Genetic Algorithm

is evolved to $\vec{T} = [2,1,2]$ which changes the deployment topologies. However, the performance of this algorithm degrades when the search space contains large number of points that corresponds to solutions that do not meet design constraints.

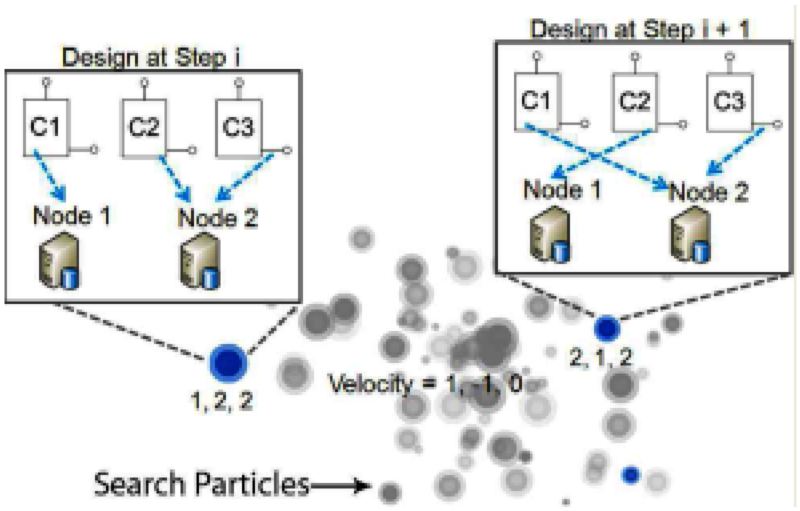

Figure VI.3: Representing a Spatial Deployment Topology as a Vector

CHAPTER VII

INTEGRATING BIN PACKING HEURISTICS WITH EVOLUTIONARY ALGORITHM

To address the challenges described in Chapter IV, we propose using a hybrid algorithm that integrates bin packing heuristics with evolutionary algorithms so that we can reap the benefits of both while overcoming the limitations of individual techniques. Moreover, rather than fixing a specific heuristic or an evolutionary algorithm, we propose to provide a framework that enables a deployment planner to strategize the framework with the desired techniques. The advantage of using bin-packing heuristics is that they produce a valid deployment topology while the advantage of using evolutionary algorithms is that they explore multiple solutions in the design space.

This chapter describes SmartDeploy, which is a strategizable framework for deployment planning that addresses the three challenges described in Section **??**. We show how the SmartDeploy framework is applied to solve the Service Uptime Maximization problem. Figure VII.1 shows the SmartDeploy framework combining worst-bin packer and PSO algorithm. It shows a generic interface to encode objective functions and constraints, and the hybrid algorithm to solve design-time constraint optimization problems. The algorithm for combining worst-fit bin packer and genetic algorithm is also similar. The white colored blocks shows the newly added features by Smartdeploy, blue colored blocks shows the integration between original and new features and the grey colored blocks show the original features of the ScatterD.

To concretely describe our solution, we use our case study example in Figure III.1. Here phones P1 and P3 are Android-based HTC phones while phones P2 and P4 are iPhones.

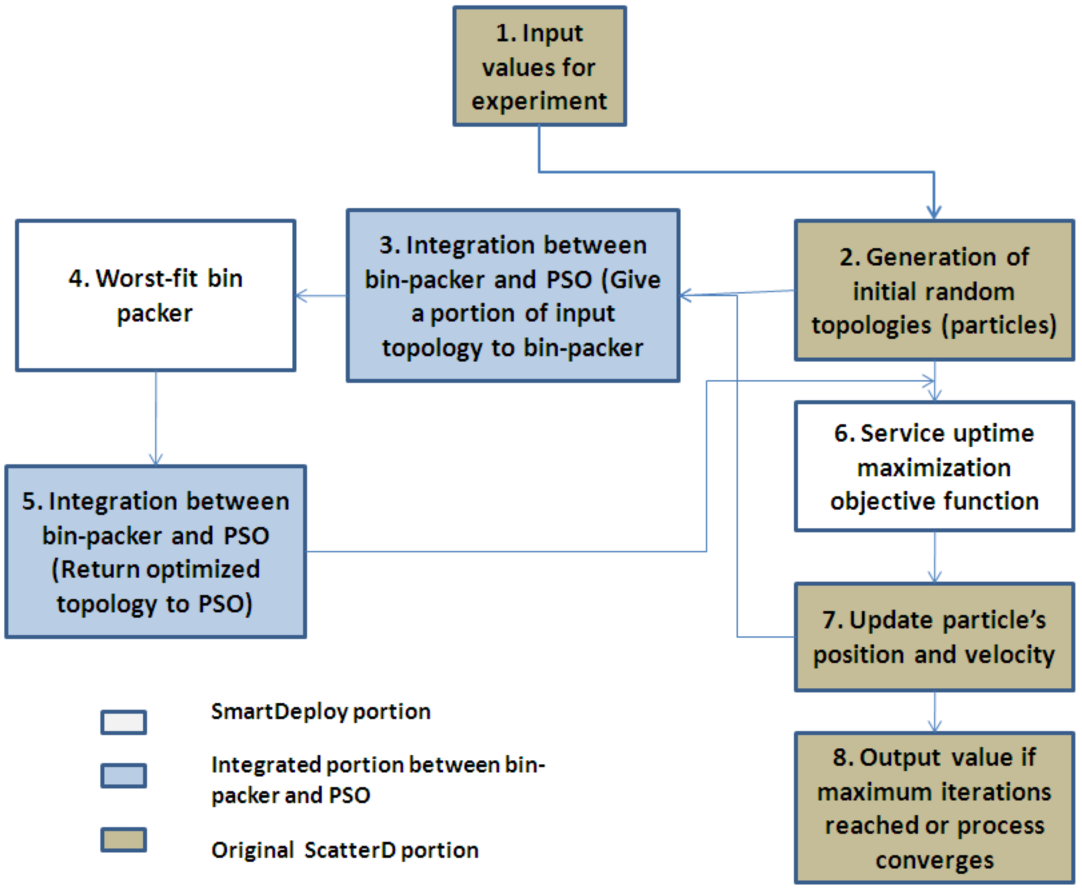
1. Input values for experiment
2. Generation of initial random topologies (particles)
3. Integration between bin-packer and PSO (Give a portion of input topology to bin-packer
4. Worst-fit bin packer
5. Integration between bin-packer and PSO (Return optimized topology to PSO)
6. Service uptime maximization objective function
7. Update particle's position and velocity
8. Output value if maximum iterations reached or process converges
SmartDeploy portion
Integrated portion between bin-packer and PSO
Original ScatterD portion

Figure VII.1: SmartDeploy framework

Software components C1, C4, and C5 can be executed only on Android-based phones while software components C2, C3, and C6 can be executed only on iPhones. The hardware and software resource requirements of the components are shown in the figure. The configuration of phones is also shown in the figure. One of the possible deployment topologies is

$\vec{V}_1 = [1,2,2,3,3,4]$.

Since there are four phones, they can run for different amount of times based on power consumption rate of the software components deployed on to them. The uptime for the phones of $\vec{V}_1$ is represented by

$\vec{st}_1 = [24, 17.1, 33.3, 25]$

which represents the index positions of the phones and the values at the index positions indicate the service uptime of the respective phones. The service uptime of the $\vec{V}_1$ is

$t_1 = Min(\vec{st_i})$ = 17.1 hours.

Here we take the minimum value from $\vec{st_1}$ as we are considering that the entire distributed application is operational only if all the phones are running. There can be many combinations of topologies like $\vec{V_2} = [1,2,4,3,1,2]$ and so on. The uptime for the phones of $\vec{V_2}$ is represented by

$\vec{st_2} = [13.3, 50, 20, 50]$ and thus its service uptime is t_2 = 13.3 hours.

The maximum service uptime of all the topologies is calculated as Max(t_1, t_2,...), which is t_1 = 17.1 hours in this case.

The more generalized formula for service uptime maximization function is defined as follows:-

$$P(\vec{V_i}) = r(\vec{V_i}) + s(\vec{V_i}) + l(\vec{V_i})$$

$$F(\vec{V_i}) = \begin{cases} e(\vec{V_i}) & \text{if } P(\vec{V_i}) = 0, \\ -1 * P(\vec{V_i}) & \text{otherwise.} \end{cases} \tag{VII.1}$$

where $r(\vec{V_i})$ is a function of resource constraints like CPU and memory. $s(\vec{V_i})$ is a function of scheduling constraint. $l(\vec{V_i})$ is the execution platform constraint, i.e., requirement of the software components to be deployed on a specific execution platform. The output of constraint functions is equal to 0 if the constraints are satisfied, else it gives the number of the constraints violated. Here $F(\vec{V_i})$ is equal to objective function $e(\vec{V_i})$ if the values of hardware resource constraint and scheduling constraint functions are 0, *i.e.*, the constraints are satisfied. Here, the objective function $e(\vec{V_i})$ is that of maximizing service uptime as explained using the case study example. Constraint functions can be added or removed as required. If the summation of constraint functions as represented by $P(\vec{V_i})$, is not 0, then an invalid topology is produced. The invalid topologies are scored on the basis of number of constraints violated. The valid topologies are always ranked higher than the invalid topologies. We need to minimize the generation of invalid initial random vectors and the evolved vectors to realize valid deployment plan. To achieve this a subset of deployment topology is sent to bin-packer. The heuristics of bin-packer has more probability to generate a valid deployment topology. However, it produces a single valid solution which is not necessarily optimized for service uptime maximization. The constraints in the bin packing algorithm are varied by a semi-random vector produced by evolutionary algorithms. Thus the evolutionary algorithms act as catalyst for exploring the solution space through semi-randomized executions of a bin packing algorithm.

A concrete manifestation of the SmartDeploy framework that combines the worst-fit heuristic bin packing algorithm with evolutionary algorithm to solve the Service Uptime Maximization problem can be described as follows:-

1. Each population member in the evolutionary search process is assigned a random initial vector, $\vec{V_i} = \vec{random}$. This is represented by blocks 1 and 2 in Figure VII.1.

2. For $i = 0$, $i < |V_i|$, a worst-fit bin-packing algorithm takes the software component referred to by position i and places it on a hardware node. The node that each component is placed on is recorded in the deployment topology vector, $T = dV_i$. The software components that are not placed on a node in Step 2 are placed into a list, L. This is represented by block 3 in Figure VII.1

3. The software components in are sorted using a bin-packing heuristic, such as memory. Each software component in L is placed on a hardware node using a standard bin-packing algorithm. Here we do not take all the components in L as it is computationally expensive. The node that each component is placed on is recorded in the deployment topology vector, dV_i. This is shown as blocks 4 and 5 in Figure VII.1

4. The score for each population member is calculated using a fitness metric as a function of the deployment plan $F(d\vec{V}_i)$, and not directly from the population member's vector, $\vec{V}_i$. This is shown represented by block 6 in Figure VII.1

5. An evolutionary operator, $evolve(\vec{V}_i)$, is applied to each population member to produce the population members for the next iteration of the algorithm. This is shown represented by block 7 in Figure VII.1

6. Steps 2-5 are repeated until either the maximum number of steps is reached or the process converges on a single solution. The highest scoring deployment topology, dV_i, is returned as the result. This is shown represented by block 8 in Figure VII.1

CHAPTER VIII

EVALUATING THE MERITS OF SMARTDEPLOY FOR SERVICE UPTIME MAXIMIZATION

This section compares the projected lifespan of an experimental mission based on its deployment plan generated by SmartDeploy PSO, SmartDeploy Genetic, PSO, Genetic and worst-fit bin packing algorithms. First we describe the experimental setup. Next we describe results of the different experiments we conducted.

VIII.1 Experimental Strategies and Execution Platform

We compared the deployments produced by five different deployment techniques. The five techniques we compared are:

1. Worst-fit bin packing - A worst-fit heuristic of bin-packing algorithm.

2. PSO - Only PSO algorithm from SmartDeploy framework.

3. SmartDeploy PSO - The PSO variant of SmartDeploy which combines worst-fit bin-packer with PSO algorithm.

4. Genetic - Only Genetic algorithm algorithm from SmartDeploy framework.

5. SmartDeploy Genetic - The genetic variant of SmartDeploy which combines worst-fit bin-packer with genetic algorithm.

The experiments were conducted on a single Windows XP desktop with 2.19 G Hz Intel Core 2 Duo processor and 2 GB RAM. Java Virtual Machine (JVM) version 1.6 was used for the experiments. For both PSO and genetic algorithm, a population size of 20, local learning coefficient of 0.5, global learning coefficient of 2, and 20 search iterations (generations) were used. The genetic algorithm allowed a total of 10% of the population to be passed through to the next generation, selected the top 25% of solutions for mating, and applied a mutation probability of 5%. A uniform distribution for generating initial random vectors is used to cover more area and not inadvertently bias our search to a specific region.

VIII.2 Experiments

Experiments 1 and 2 described below were conducted using 100 nodes and 100 software components. The number of nodes tested for the experiment ranges from 30 to 100. The number of software components are kept constant.

Experiment 1: Homogeneous nodes, heterogeneous software components –

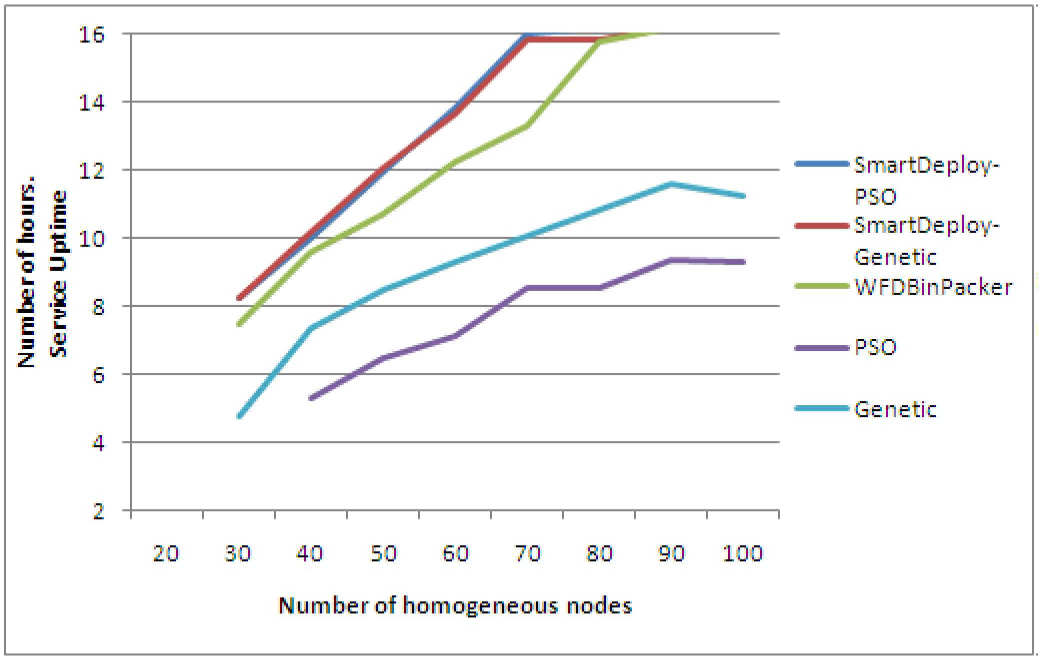

Figure VIII.1: Homogeneous nodes, Heterogeneous software components

The first experiment was conducted using homogeneous nodes, *i.e.*, each of them having the same amount of memory and power capacity on them. The software components deployed on them were heterogeneous, *i.e.*, each of them requiring different amount of memory and power consumption capacity. Here the constraints based on the amount of memory available on all nodes and the amount of memory required by all the software components are used, *i.e.*, total hardware and software resource requirements should not exceed their total availability.

Hypothesis SmartDeploy should provide significant increase in service uptime compared to the bin-packing algorithm and PSO. Here although the nodes have homogeneous properties for the amount of memory and the battery power capacity, the heterogeneous properties of the software components *i.e.*, each of them requiring different amount of memory and power consumption capacity causes SmartDeploy to produce better results than the worst-fit bin packer and and evolutionary algorithms alone.

Analysis of results As seen in the Figure VIII.1, SmartDeploy algorithms show 94% and 58% improvement in maximizing service uptime over PSO and genetic algorithms, respectively. However, it gives only 20% improvement over worst-fit bin packer. After careful analysis, it can be seen that due to the homogeneous properties of the nodes, the worst-first bin packer gives better results as compared to both the evolutionary algorithms, and are close to that of SmartDeploy.

Experiment 2: Heterogeneous nodes, heterogeneous software components –

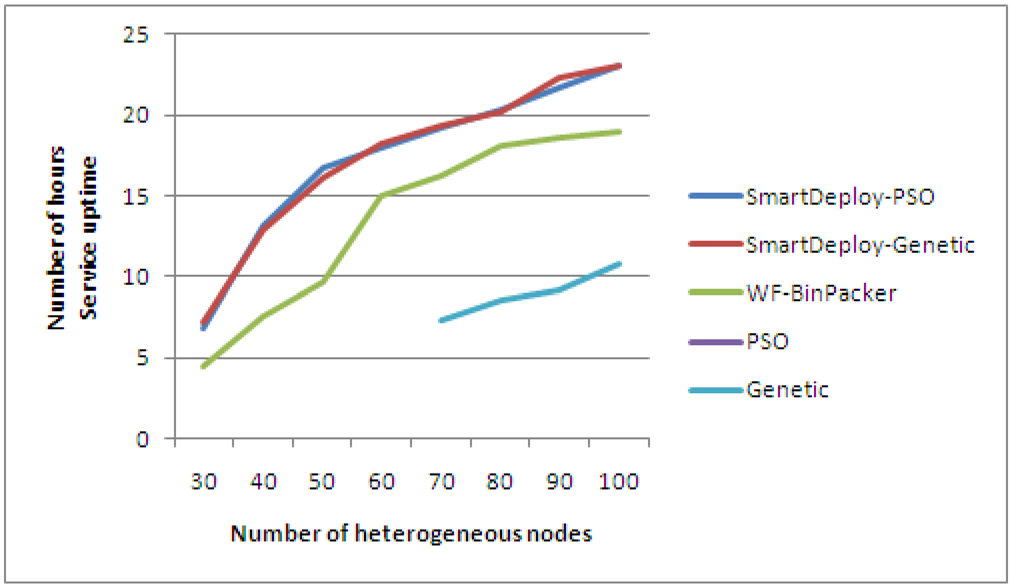

Figure VIII.2: Heterogeneous nodes, Heterogeneous software components

The second experiment was conducted using heterogeneous nodes, *i.e.*, half the number of nodes have one set of properties while the other half have another set of similar properties. For lack of space we do not report on other variations. The software components deployed on them were also heterogeneous, *i.e.*, each of them requiring different amount of memory and power consumption capacity. Here the constraints based on the amount of memory available on all nodes and the amount of memory required by all the software components were used, *i.e.*, total hardware and software resource requirements should not exceed their total availability.

Hypothesis SmartDeploy should provide significant improvement in service uptime compared to the bin-packing algorithm and evolutionary algorithms. Here the nodes having heterogeneous properties for the amount of memory and the battery power capacity, the heterogeneous properties for the software components, *i.e.*, each of them requiring different amount of memory and power consumption capacity should cause the SmartDeploy algorithms to produce better results than the worst-fit bin packer and evolutionary algorithms alone.

Analysis of results As seen in Figure VIII.2, due to the heterogeneous properties of nodes and software components, and large problem size, the performance of evolutionary algorithms degrades. PSO gives invalid topologies in this scenario. Genetic algorithm gives invalid topologies when software components are tightly packed onto devices. Even when the number of devices increases, SmartDeploy algorithms provide up to 162% better service uptime. They also provide up to 75% more service uptime than worst-fit bin packer.

Experiment 3: Varying the number of software components (heterogeneous) deployed on fixed number of heterogeneous nodes –

The third experiment was conducted by varying the number of heterogeneous software components being deployed on fixed number of heterogeneous nodes. The number of software components varied from 100 to 200 with increments of 20. Here the constraints were based on the amount of memory available on all nodes and the amount of memory

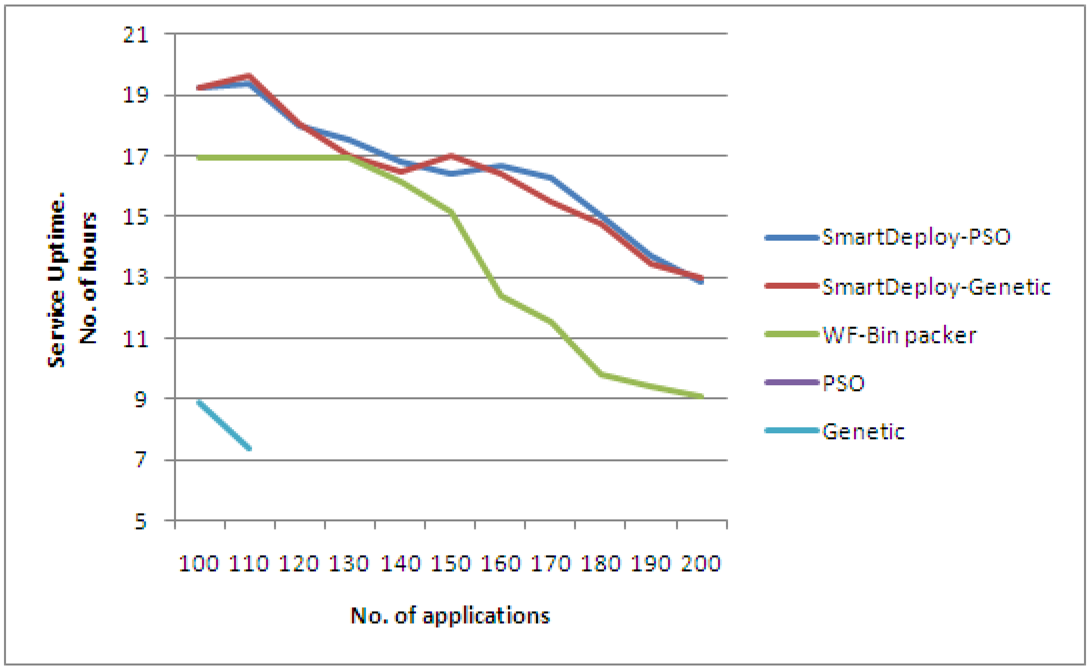

Figure VIII.3: Varying the number of software components(heterogeneous)

required by all the software components are used, *i.e.*, total hardware and software resource requirements should not exceed their total availability.

Hypothesis As the number of software components increases, the topologies become tightly constrained. If the solution space increases, then it should cause the bin-packer to provide a less than optimal value. The tightly constraint solution space should cause evolutionary algorithms to degrade in their performance.

Analysis of results As seen in the Figure VIII.3 the devices become tightly packed with increasing number of software components and constraint on memory requirements. The evolutionary algorithms degrade in performance and give invalid deployment topologies. The SmartDeploy algorithms give up to 50% more service uptime as compared to worst-fit bin packer.

Experiment 4: Heterogeneous nodes (different OS) and heterogeneous software components –

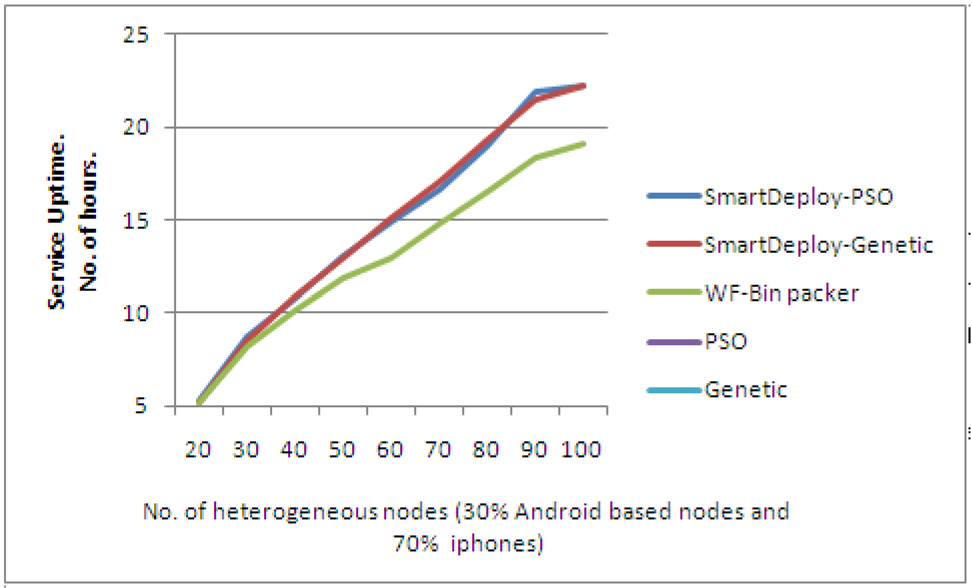

Figure VIII.4: Heterogeneous nodes(different OS), Heterogeneous software components

The fourth experiment was conducted using heterogeneous nodes, *i.e.*, 30% of nodes have one set of properties while the 70% of nodes have another set of similar properties. Also, different OS (Android-based and iphone) was used for either set. The software components deployed on them were also heterogeneous, *i.e.*, each of them requiring different amount of memory and power consumption capacity and execution platform(OS). Here the constraints based on the execution platform (OS), amount of memory available on all nodes and the amount of memory required by all the software components were used, *i.e.*, total hardware and software resource requirements should not exceed their total availability.

Hypothesis SmartDeploy should provide significant improvement in service uptime compared to the bin-packing algorithm and evolutionary algorithms. Here the nodes having heterogeneous properties for the amount of memory, the battery power capacity and execution platform(OS), the heterogeneous properties for the software components, *i.e.*, each of them requiring different amount of memory and power consumption capacity and execution platform(OS) should cause the SmartDeploy algorithms to produce better results than the worst-fit bin packer and evolutionary algorithms alone.

Analysis of results As seen in Figure VIII.4, due to the heterogeneous properties of nodes and software components, and large problem size, the performance of evolutionary algorithms degrades. PSO gives invalid topologies in this scenario. Genetic algorithm gives invalid topologies when software components are tightly packed onto devices.SmartDeploy algorithms give higher service uptime than bin-packer and the evolutionary algorithms.

Experiment 5: Comparison of service uptime by all the algorithms with that of brute-force algorithm –

Table VIII.1: Time taken to run Brute-force algorithm for service uptime

Nodes	Software components	Service uptime(m sec)
5	5	78
5	7	1219(1.2 secs)
5	9	33312(33.3 secs)
5	11	1261211(21 minutes)

We attempted to obtain the optimum service uptime using brute-force algorithm which tries each and every combinations of deployment topologies. However, we observed that running the brute-force algorithm even for even small problem sizes takes significant time. So it was not practical to run it for large problem sizes of hundreds of nodes and hundreds of software components. Table VIII.1 shows the running time for brute-force algorithm over a small problem size.

Experiment 6: Comparison of computation time taken by each of five algorithms to execute –

The sixth experiment was conducted to observe the average time taken by each of the five algorithms to execute. Here the experimental values used in experiment 2 were used, *i.e.*, heterogeneous nodes and heterogeneous software components. The average values for

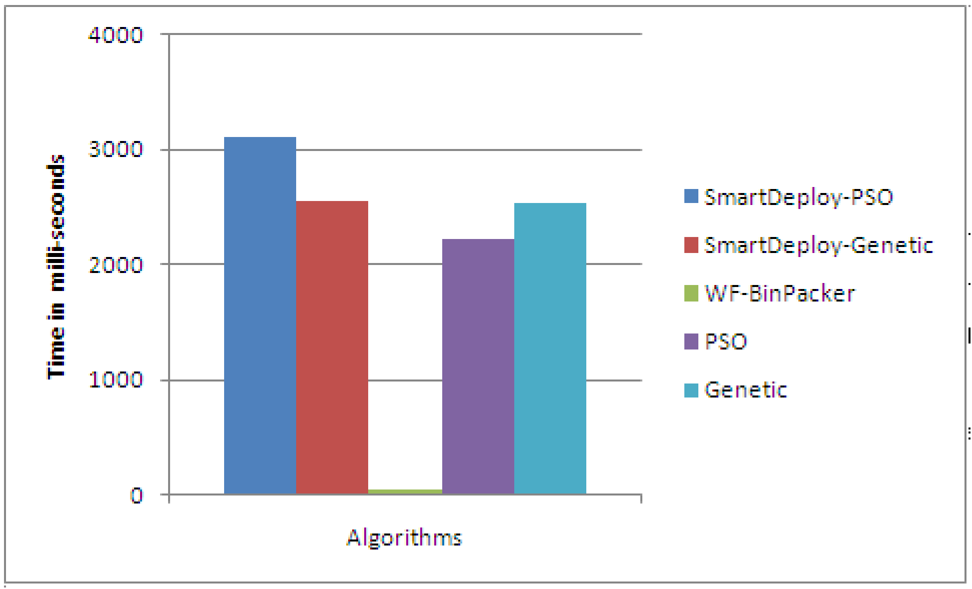

Figure VIII.5: Time taken by each of five algorithms to run

service uptime for the entire range of nodes were taken. As seen in Figure VIII.5, worst-fit bin packer takes least amount of time to run, *i.e.*, 47 milliseconds. The SmartDeploy algorithms take most amount of time to run, *i.e.*, between 2,000 milliseconds to 3,200 milliseconds. Since we are considering an offline solution for deployment topology, a delay in few seconds is tolerable to achieve better service uptime. Hence the use of SmartDeploy algorithms is desirable in such situations.

CHAPTER IX

CONCLUDING REMARKS AND LESSONS LEARNED

Service uptime maximization in distributed applications hosted on a network of smartphones can be achieved through effective deployment. Several optimization techniques are commonly used for deployment problems in distributed real-time and embedded (DRE) systems. Algorithms with exponential runtime complexity like integer programming are not scalable when the problem size increases up to hundreds of devices. Bin-packing heuristics tend to generate valid deployment topologies, but they may not give optimal solutions when problem size increases. Evolutionary algorithms are commonly used for deployment problems since they explore a variety of design solutions. However, as the number of constraints and the problem size increases, they tend to degrade in performance.

The thesis described a framework called SmartDeploy that provides a hybrid deployment technique to achieve service uptime maximization. It builds upon the earlier work done in the group, called ScatterD, which combines first-fit bin packer with the evolutionary algorithm to reduce power consumption in DRE systems. SmartDeploy enables a user to strategize both the evolutionary algorithm as well as the bin packing heuristic. A concrete manifestation of SmartDeploy using the worst-case bin packer along with evolutionary algorithms is presented to solve the service uptime maximization problem for smartphone-based mission critical applications.

Using worst-fit bin packer heuristic, the software components of the distributed application can be evenly deployed on the available devices such that they can obtain maximum available battery power and sufficient hardware resources. The experimental results show that SmartDeploy framework increased service uptime from 20% to 162% beyond that provided by worst-fit bin packer and evolutionary algorithms used independently. The following lessons were learned conducting this research:

- Since the running time of the SmartDeploy algorithms is only slightly more than the algorithms we compared against, it is practical to use the hybrid algorithm. In future work we intend to investigate the use of SmartDeploy framework in runtime deployment decisions.

- We also intend to investigate other distribution techniques for generation of initial random topologies of evolutionary algorithms like Gaussian distribution to see if they can achieve better solutions.

- We intended to run the brute-force optimal algorithm to compare the service uptime solutions from each of the five algorithms we used in our experiments to see how our solutions compare to the optimal one. However, we observed that running the brute-force algorithm even for small problem sizes takes considerable amount of time. Hence it was not practical to test it out for the large problem size that we use.

REFERENCES

[1] http://spie.org/x648.html?product_id=850347.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA, 1996.

[3] M.C. Bastarrica, A.A. Shvartsman, and S. Demurjian. A Binary Integer Programming Model for Optimal Object Distribution. In *2nd Int. Conf. on Principles of Distributed Systems*.

[4] Brian Dougherty, Jules White, Jaiganesh Balasubramanian, Chris Thompson, and Douglas C. Schmidt. Deployment Automation with BLITZ. In *Emerging Results track at the 31st International Conference on Software Engineering*, Vancouver, CA, May 2009.

[5] JN Hooker. Planning and scheduling by logic-based benders decomposition. *OPERATIONS RESEARCH-BALTIMORE THEN LINTHICUM-*, 55(3):588, 2007.

[6] Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhante. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[7] Andrew Howard, Maja J Mataric, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. pages 299–308, 2002.

[8] Jules White and Brian Dougherty and Chris Thompson and Douglas C. Schmidt. ScatterD: Spatial Deployment Optimization with Hybrid Heuristic / Evolutionary Algorithms. *ACM Transactions on Autonomous and Adaptive Systems Special Issue on Spatial Computing (to appear)*, 2010.

[9] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, 1995.

[10] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[11] FranÃğois Laburthe, Narendra Jussien, Hadrien Cambazard, and Guillaume Rochart. choco: an open source java constraint programming library.

[12] Charles Perkins and Elizabeth Royer. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1997.

[13] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *ACM Conference on Communications Architectures, Protocols and Applications, SIGCOMM '94, London, UK*, pages 234–244. ACM, ACM, August 1994.

[14] R. Simmons, D. Apfelbaum, D. Fox, RP Goldman, KZ Haigh, DJ Musliner, M. Pelican, and S. Thrun. Coordinated Deployment of Multiple, Heterogeneous Robots. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, 2000.

[15] Wu Xiaoling, Shu Lei, Yang Jie, Xu Hui, Jinsung Cho, and Sungyoung Lee. Swarm based sensor deployment optimization in ad hoc sensor networks.