SENSING AND ACTUATING TASKS AS SERVICES AND ITS QUALITY OF SERVICES IN LARGE CLUSTERED ENVIRONMENTS

By

THRISHUKANTH DASARI

Bachelor of Engineering in Computer Science

Osmania University

Hyderabad, AP, India

2007

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2011

SENSING AND ACTUATING TASKS AS SERVICES

AND ITS QUALITY OF SERVICES

IN LARGE CLUSTERED ENVIRONMENTS

Thesis Approved:

ACKNOWLEDGMENTS

It gives me an immense pleasure to thank everyone involved in the successful completion of my thesis.

It is an honor for me to work under the supervision of Dr. Johnson P. Thomas, and I would like to take this opportunity to thank him for his invaluable guidance and support throughout the thesis. For all his help, knowledge, and patience, one could not wish for a better or friendlier advisor.

I would also like to thank the members of my committee, Dr. Subhash Kak and Dr. Michel Toulouse for their time, support, and advices which helped me improve and complete my thesis. I also thank Mr. Russ Smith and Ms. Dana Brunson for supporting me with the infrastructure needed for the simulation of my work.

I feel very proud and lucky to be a part of my loving and supporting family, and I would like to thank my parents, brothers, sister, and in-law's for their encouragement and belief in me. Without their support, I would not have achieved anything.

Finally, I would like to thank Prerana Laddha, Soujanya Vullam, Vishnu, Varsha, Karthik, and Swathi for their endless support and friendship

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1 Wireless Sensor Networks

Advances in distributed computing, sensing, and wireless communication technologies led to the emergence of wireless sensor networks. *Sensors* observe physical and environmental conditions such as sound, light, temperature, pressure, motion, etc. and communicate the observed information amongst each other or to a remote application. These sensor nodes are spatially distributed to form a Wireless Sensor Network (WSN). Sensors communicate using wireless links and vary in size, speed, and bandwidth. These sensing and communicating capabilities are used in various applications such as healthcare monitoring, environment and habitat monitoring [8], building surveillance, home automation, business applications, and entertainment [4].

1.2 Wireless Sensor and Actuator Networks

Monitoring of a physical system remotely along with real-time controlling has become a necessity in recent times, facilitating rapid improvements in wireless sensor networks. Thus, the course of taking a decision and actuating an action in a physical system, depending on the sensor’s information, has become functional in the emerging mission critical applications, such as an intelligent sprinkler system, autonomous animal control, environmental control, home automation, event detection and suppression, manufacturing, cyber physical systems etc. This emergence of manual initiated and/or sensor-triggered automatic actuation in controlling the physical system, in the form of *Actuators,* laid the foundation for *Wireless Sensors and Actuator Networks (WSANs).* Actuators perform actions to change the parameters of the environment or physical systems and have strong computational and communication powers with high energy and long battery-life, compared to sensors [1, 2]. Sensors and actuators are interconnected over wireless links, facilitating distributed interactions with the physical world [4]. In this dissertation, the terms WSANs and SANETs (Sensor and Actuator Networks) are used interchangeably.

WSANs constitute a large pool of easily usable and accessible sensor/actuator resources, enabling communication in the network to dynamically reconfigure and perform distributed sensing and actuation tasks. For example, an intelligent traffic controller monitors congestion on the road and automatically routes upcoming traffic using electronic display boards to take appropriate actions and detours. The control flow of the sensors and actuators in SANETs is explained briefly in the next subsection.

1.2.1 Control flow for performing sensing/actuating tasks in WSANs:

SANET resources can be used to sense, interact, and affect the behavior of physical systems [6] by enabling different ways of performing sensing and actuating tasks, according to the requirements. The closed loop control flow in a generic architecture of a WSAN is shown in the figure 1.

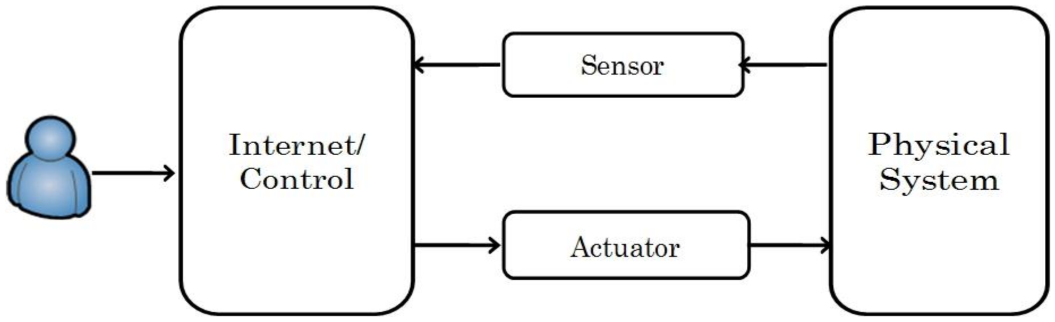

Figure 1: Closed loop control flow in WSANs

When a user initiates a task, the feedback from the sensor, i.e.; the information about the specific physical parameters sensed, is sent to the remote controller over the network. The controller enables the actuators to adjust the physical system parameters to meet an output response requested by the end user. For example, a user automating the system to turn the lights off if nobody is present in the room. The second is a manually initiated actuation, where the actuation is initiated by the user without depending on feedback from sensors about physical system parameters. For example, a user requesting to turn the lights off irrespective of the presence of people inside the room. The third is an example of WSNs with just the sensors monitoring the physical system. The sensor monitors the physical system and gets the output parameters to the user. For example, a user requesting to know the number of people present in the room or a doctor requesting to know the blood pressure of the patients.

1.3 Motivation and Research Objective

We envisage the world quickly advancing to an era with a connection of millions of Sensing and Actuating resources everywhere, performing tasks in day-to-day life. The ability to trigger these tasks without worrying about the ownership and availability of the resources as well as the process of searching and scheduling, resources, given the location of the user himself, is provided by enabling *Sensing and Actuating Tasks* as services. Cloud computing will become more dominant than the desktop in the next decade [14], motivating us to prospect a quick development in SANETs, to facilitate Sensing and Actuating resources on demand via a computer network. Thus, we propose an architecture called SATS (Sensing and Actuating Tasks as Service) with an ability to trigger Sensing and Actuating Tasks using the Internet, to perform cyber-physical tasks. The SANETs are owned, maintained, and used by groups such as private providers, research institutions, government organizations [4]. Each party offering SANET resources as services is named as a SANET Service Provider (SP). There could be many such Service Providers in the market, owning millions of sensing and actuating resources.

A SP might own a massive number of Sensor and Actuator resources within a number of SANETs. The distributed resources from different SANETs communicate with each other using multi-hop wireless/wired links. The services are published on the web with specific details and protocols to guide the SANET Service Requester's (SR) use. This enables SRs to access services using a web browser regardless of their location and device (e.g., PC, mobile). The services provided by SPs are distributed globally and a service offered by one SP may require a series or combination of execution of multiple services provided by other SPs. Thus, it is important that these services are interconnected and interoperable to provide the desired services to the SRs. These interoperable services offered by the Service Provider, enable the Service Requestor to place a request to perform an activity from anywhere in the world using an internet connection. Handling the request involves either simple data passing or multiple services coordinating some activity. The resources involved in coordinating an activity may not belong to the same SP. Each SP may have multiple resources that perform the same task, depending on the demand of the tasks at that specific location (e.g. an SP owning more than one temperature sensor in the same room to find its temperature, or an SP owning multiple sensors to run user code for research purposes). Two or more SPs may also have similar resources in common, which can perform same tasks and are available at the same location (e.g. two SPs having multiple temperature sensors in the same room to find temperature). This provides a possibility of choosing an idle resource among all the available resources which can perform the required task. The ability to raise a request using the web, and choose a resource among multiple such resources which might also be owned by different SPs, primarily has the following advantages

- ✓ Optimal and maximal utilization of available resources.
- ✓ Decreases waiting time of the request by allotting an idle resource, instead of waiting for a busy resource to become idle.
- ✓ Reduces cost by saving infrastructure, as an SP can choose an idle resource owned by another SP, to perform the task.
- ✓ Increases flexibility for the SP, as all the possible resources that are available to perform the task is known, thereby decreasing the probability of "Denial of Service".
- ✓ More mobility to the users, who can raise a request and access the resources regardless of their location.
- ✓ Increases interoperability amongst the resources.

1.4 Challenges

SANETS face a number of challenges as mentioned below.

Timing Constraint: SANET may require real-time task execution as the sensor data may have to be acted on by the actuators within timing constraints. So, scheduling the execution of tasks is a challenge.

Search Space: As there may be millions of sensors and actuators, the search space is also a problem.

Resource Conflict: There may be resource conflicts as multiple tasks may need to access the same resource.

Fault tolerance: It is important as sensors in particular are liable to failure due to their limited resources.

Cost effective: As the SANET resources are available as services; the selection of cost effective resources amongst multiple available resources is a challenge.

As these systems deal with physical phenomenon, geographic location of sensors and actuators may be relevant.

1.5 Research Contribution

Our proposed SATS architecture focuses on the selection of the best possible resources available in the market to perform the request raised by the user in a minimal time to meet task timing requirements. In this process, we propose a Resource Selection (RS) algorithm to address the challenges specific to resource selection, as mentioned in last subsection.

Searching for idle resources in a large clustered distributed environment with massive number of resources and their data, necessitates processing and generating data sets on large clusters. In the RS algorithm, we use the Map-Reduce framework [13] that simplifies data processing on large clusters by processing and generating large data sets in an efficient way. It is a programming model with the implementation of *map* and *reduce* functions. The Map function processes the key/value pair to generate a set of intermediate key/value pairs, and the Reduce function processes the intermediate key/value pairs generated by the map function to generate the output by merging all intermediate values associated with the intermediate key [7].

The SATS architecture has three levels of communication. The sensing and actuating services are provided by the Service Providers as an interface where the user has to fill in the details of his area(s) of sensing or/and actuating tasks to be done. The request raised by the SR is passed to the SP (local SP) components with the execution flow and the details of the request. The local SP broadcasts a Search Request to all the other SPs (Foreign SPs) in the market and gets the response from each SP with the available resources owned by each of them, which is the output after applying the RS algorithm. Thus, the local SP will have all the available idle resources in the market that can be utilized in performing the requested task from each SP. As the idle resources returned from two or more SPs may have the same functional features, the local SP selects the number of resources required from each set of similar resources, and are ranked to find the optimal combination according to their cost of usage. As all the requests are processed in parallel, there could be multiple requests on similar or same resources during the time of the whole resource selection process. Thus, the availability of the selected resources is checked again using the local registry information. This check is performed on all the ranked resources of each set, sequentially, until an available resource is found. The final selection of resources generated after applying RS algorithm is time and cost effective, location confined, and schedule specific.

1.6 Organization of the Thesis

The rest of the dissertation is organized as follows. Chapter 2 provides background of the research done in the field of WSNs and WSANs related to different challenges and architectures proposed in the communication, control, coordination, virtualization, and selection areas of the SANET resources. Chapter 3 provides the architecture for the interoperability amongst the SPs, facilitating resource virtualization and selection to perform the tasks requested by the user. Chapter 4 provides the implementation of the RS algorithm on millions of resources owned by different SPs in the market. Chapter 5 provides the conclusion. And, chapter 6 provides the future work.

CHAPTER II

REVIEW OF LITERATURE

Over the past few years, research has been done on enhancing applications and architectures involving Sensors and Actuator networks.

2.1 Service Oriented Sensor and Actuator Network

The SOSANET [4] approach builds service based customizable SANETs to handle the given applications. It claims that most approaches are either application-specific or generic. Application-specific SANETs provide limited reusability with cost ineffectiveness and require reprogramming to make it useful for the application, resulting in tight coupling between the application and the underlying SANET, and increased energy efficiency and scalability. Generic SANETs require specific code be deployed on nodes without having pre-knowledge about the applications, resulting in decoupling of the application and underlying SANET, and the chance of code reusability. SOSANET provides the benefits of both application-specific, and generic SANETs. Customizable SANETs provides flexibility to combine the resources provided by nodes in one or more SANETs to meet the application requirements. This approach uses generic SANETs as backbone along with an additional software layer on each node which provides some functionality.

They deploy services directly on top of the operating system, and services are accessible directly by applications. It uses the Service Driven Routing (SDR), where each node apprehends the other node's potencies in providing their services. Each node has a Service Directory that stores all the general and availability information about the services provided by its reachable nodes. So, when a request is made, each node generates a query result and sends it to its neighbours. Each neighbor forwards it until it reaches the base station. The results show that it had significant improvements over existing architectures in energy consumption, scalability, and response time. The request query in the SOSANET approach is initiated at the node, thereby passing the result to neighbors till it reaches base station. This works well for a limited number of nodes and requests at each instance. As the author says, future SANETs require a new architecture. The disadvantage with this architecture is passing the result to neighbors, as this increases the communication on the wired links, resulting in message overheads, affecting the communication capabilities and energy. While SANETs will become undoubtebly ubiquitous in the future, with millions of users wanting to use the resources at the same time, it might face performance issues as for each request, each node tries to forward its reply to all its neighbors in a huge network of interconnected SANETs distributed all over the world. This may also lead to synchronization issues. The handling of such a massive network of SANETs owned by disparate SPs requires to reduce the lowest level message passing, amongst the energy dependent sensors and actuators. SATS archtiecture reduces the message passing by filtering all the resources, except the ones which are idle and have the capabilities to perform the task, that is, in the first step itself. SATS is based on the network with a large number of Sensing and Actuating resources, and the SPs owning them. The SOSANET appraoch stores the service directory at each node wasting its storage and capacity, whereas SATS stores local registry at each controller of all the WSANs, thereby increasing the capabilities of the resources. This is imporant as sensors have limited resources.

2.2 Sentire middleware

Sentire [5], a high-level application middleware, emphasizes the control and coordination of time dependent and time sharing SANET resources using market-based bidding strategies. This framework supports large-scale distributed SANETs with a design of development tools for providing reusable model and actuator coordination techniques to reduce task interference due to the integration of large-scaled SANETs. This minimizes the damage to the physical world due to the use of shared resources, and performs uncoordinated actions with multiple actuators being close to each other, while performing a task. Sentire implements a market-based algorithm for resource allocation to coordinate the distributed actuators. The agents continuously bid against each other for the common actuators in the market, and wins access to use common resources. Hence, each agent is aware of the other agent's bid and each agent is aware of the next user of the shared resource. This reduces the task interference and selects the resource by bidding.

2.3 Handling mobility in Wireless Sensor and Actuator Networks

Akyildiz [3] proposed Distributed Event-driven Partitioning and Routing (DEPR), a distributed protocol for sensor-actor coordination that includes an adaptive mechanism to trade off energy consumption for delay when the event data has to be delivered to the actors within predetermined latency bounds. For the actor-actor coordination, an optimization model was defined for a class of coordination problems in which the area to be acted upon is optimally split among different actors. The problem was formulated as a Mixed Integer Linear Program (MILP) and an auction-based localized solution of the problem was also presented. They extended their work by proposing a new location-management scheme, which combines joint use of Kalman filtering and Voronoi scoping on the sensors and actors, for efficient geographical routing of sensor-actor communication. It also proposes actor-actor coordination, coordinating the motion and the action of the participating actors by selecting the best actor team that will cause minimal reconfiguration of network operations, based on the characteristics of the event. A model is proposed to optimally assign tasks to actors and control their motion in a coordinated way to accomplish the tasks based on the characteristics of the events. The selection of a team of actuators to optimally divide the task, coordinate, and perform while respecting the action-completion bound and low movement energy to complete the task is stated as a Multi-actor Allocation problem. The sensors that generate the parameters define event area, and the actuators that perform some actions define action area. These areas may coincide in several applications. This approach mainly depends on three factors, congestion factor, directivity factor, and distance factor, to formulate a Mixed Integer Non-Linear Program (MINLP) and find the best actor team. Our approach is based on the sensor and actuator tasks, offered as services, using a trivial way of finding the best resources in large-scale distributed SANETs involving billions of resources world-wide.

2.4 Map-Reduce Framework

We use the basic idea of the successful and popular Map-Reduce framework proposed by Google Inc [7]. Map-reduce make the data processing simplified on large clusters by processing and generating large data sets. It is a programming model with the implementation of *map* and *reduce* functions. Map function processes the key/value pair to generate a set of intermediate key/value pairs, and Reduce function processes the intermediate key/value pairs generated by the map function to generate the output by merging all intermediate values associated with intermediate key [7].

$$\text{Map}\ (k1, v1) \rightarrow \text{list}(k2, v2)$$

$$\text{Reduce}\ (k2, \text{list}(v2)) \rightarrow \text{list}(v2)$$

The input key K1 and values V1 are drawn from different domain than the output keys and values, and the intermediate keys and values are from same domain as the output keys and values. MapReduce enables automatic parallelization and distribution of large-scale computations with high performance on large clusters. Partitioning of the input data, scheduling the program across a set of machines, handling machine failures, and managing inter-machine communications are handled by the real-time system. The implementation is highly scalable and has a capacity to process many terabytes of data on thousands of machines.

2.4.1 Pseudo code of Map and Reduce functions for word count

The pseudo code for the popular counting the number of occurrences of each word in a large collection of documents as given in [7] is:

```
map(String key, String value):
    // key: document name; value: document contents
```

```
        for each word w in value:
        EmitIntermediate(w, "1");
    reduce(String key, Iterator values):
        // key: a word; values: a list of counts
        int result = 0;
        for each v in values:
        result += ParseInt(v);
        Emit(AsString(result));
```

2.4.2 Examples of Map-Reduce framework

The authors also mentioned few examples such as Distributed Grep, Count of URL access Frequency, Reverse Web-Link Graph, Inverted Index, and Distributed Sort, that can be expressed as MapReduce computations. The proposed SATS architecture makes use of the concept of the Map Reduce framework in the Inverted Index example. The map function parses each document, and emits a sequence of (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (word; list(document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions. SATS also uses the optional Combiner function mentioned by the authors that does the partial merging of data before it is sent over the network. The combiner function is executed on each machine where the map function is performed. Combiner and reduce functions use the same code to implement. The output of the combiner is written into the intermediate file that will be sent to a reducer function. This partial combining speeds up certain classes of Map-Reduce operations.

CHAPTER III

PROPOSED WORK

3.1 Goal of SATS

The main focus of this chapter is to present SATS as an architecture that can efficiently handle the requests raised by the SRs to initiate a sensing or actuating task, or a group of sensing and actuating tasks to perform an activity. As we are dealing with very large-scale distributed SANETs owned by various SPs, and each SP in the market competes to have its own sensor and actuator resources where they are in demand, there could be a number of resources with similar characteristics and abilities to perform same task and are available at same location. The SR's request is performed by a group of sensors and/or actuators by dynamically reconfiguring as per the requirements of the user. The goal of SATS is to guide the process of using sensing and actuating tasks as services with the following contributions:

- A high-level framework consisting of reusable managers that facilitate the flow of user requests within a SP instance, and data between applications and underlying SANET resources used.

- Resource hiding, which protects the information about the underlying sensors and actuators used in executing the task, from the users. This makes the approach

- application independent with a dynamic reconfiguration of the resources on each request.
- Parallelized and distributed computations are provided with the use of RS algorithm, to find the idle resources from a huge network of interoperable SANETs in the market having multiple resources of similar characteristics at the same location, which can perform the user specified task. Hence, the proposed approach selects the optimal group of sensors and/or actuators, and performs a task or an activity as per the user requirements. This programmable approach can be automated by the SPs to handle the sensing and/or actuating requests raised by the user in the widely distributed SANET.

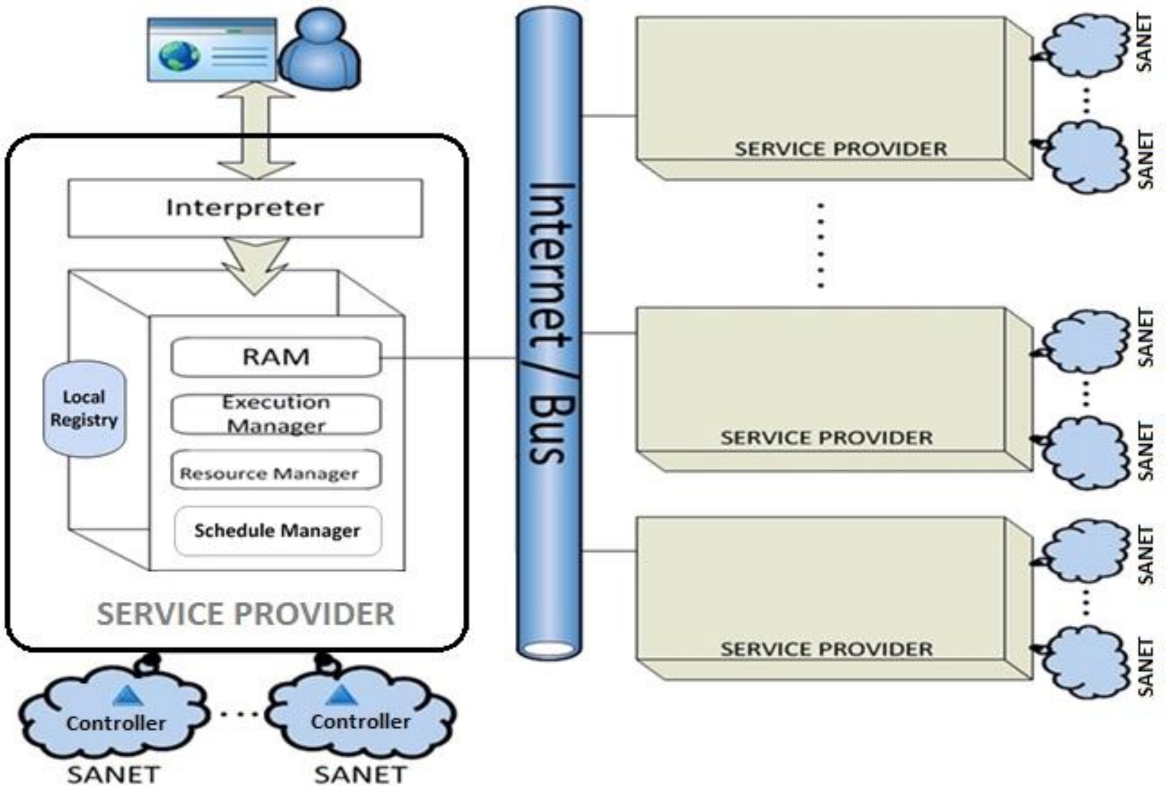

Figure 2: SATS architecture for making sensor and actuating tasks as services

Communications in the SATS mainly takes place by its set of extensible managers as shown in figure 2. The figure shows the passing of request from the user to the underlying SANET resources through the internal and external components of the SP. SATS uses publish/subscribe messaging pattern in implementing the communication among the managers. Using a publish/subscribe pattern, senders (publishers) do not sent the messages directly to specific receivers (subscribers). Instead, messages are characterized into classes, and subscribers that show interest in one or more classes, receive the messages from the subscribers they are interested in.

The problems specific to the resources are as follows:

1) The requests raised by the user are to be performed without any delay in order to meet the deadlines. As there are millions of resources available in the market owned by disparate SPs, the selection of the resources is time-intensive as the SPs have to search for the available resources. Hence, searching each SP's huge database and deciding on the list of available resources in negligible time is a challenge.

   In our work, we solve this problem by using the RS algorithm based on the Map-Reduce framework, thereby making use of parallel processing resulting in an increase in the computational slave nodes that process the data.

2) Cost, levied on the usage of selected resources, is one of the important factors before selecting. The resources selected should be cost effective.

   In our work, we solve this problem by maintaining the cost of usage of each resource within the list of available resources in the selection process. Ranking is applied on the cost, and the resource with the least cost (first in the ranked list) is selected to participate in an activity to perform the task.

3) While the resource allocation gives the available resources to do the tasks, the system should also make sure that the schedule is met given the available or selected resources. Thus, timing constraints becomes one of the important factors in resource selection and should be satisfied.

   In our work we solve this problem by maintaining the scheduled events or timings of each resource in the local registry of each SP. These are the timings at which the resource is to be triggered as scheduled by some previous request.

4) There is a possibility of service denial which could be caused due to unavailability of the resources requested at a specific location because the resource might be in use or too far away from the actual location of user's interest. Thus, the flexibility to search the resources which are close to the specified location is to be provided to decrease the probability of service denial. Hence, the location confinement problem should be addressed.

   In our work we solve this problem by calculating the locations which are within the proximity range of 'p' units from the actual location requested by the user. The resources in these locations are also considered during the resource selection process.

5) As the resources are subject to failure, the system has to have back-up resource information without having to search the SANET again for the set of available resources. Thus, failure tolerance should be addressed

In our work we solve this problem by maintaining a list of backup resources, i.e. the resources with the same order of ranking.

6) As there could be many requests processing at the same time, more than one request may need the same resource in non-overlapping time slots. Thus, the scheduled time and the requested time slot is also important in allocating the resources to maximize the utilization of each resource. Hence, the resource selection process should also address this problem.

   In our work we solve this problem by rechecking the scheduled activity timings of the selected resource, before sending its information to the Execution Manager to carry out the activity or update its scheduled timings.

The functionality of each component and the sequence of communication flow amongst each component of the SPs are described briefly in the following sections:

3.2 SATS Components

3.2.1 Interpreter Manager (IM)

The IM serves as an interface between user request and the SP instance. IM is assumed to divide the request into sub-tasks, if it involves a combination of multiple services or tasks that could be run in parallel, and gives the logical execution plan with the flow of tasks as per the requirements.

3.2.2 Resource Manager (RM)

The RM keeps track of the resources in the SANETs belonging to its local SPs, and updates their information in the Local Registry to keep it up-to-date. It is responsible for having the latest availability and other information of the resources owned by its SP.

3.2.3 Local Registry (LocReg)

The LocReg is a dataset with all the information about the sensing and actuating resources owned by each SP. The following information is stored in local registry for each resource owned by its SPs:

- ✓ *resourceId* – Unique ID assigned to the resource.
- ✓ *resourceType* – The type of resource (ex. Temperature Sensor)
- ✓ *location* – The geographical location of the resource.
- ✓ sanetID – Unique ID assigned to each SANET.
- ✓ socketID – Socket address of the resource.
- ✓ *activeFlag* – The active status flag of the resource. Y for Active, N for inactive
- ✓ *availabilityFlag* – The availability status flag of the resource. Y for free, N for busy
- ✓ *cost* – The cost of usage per unit time, charged when the resource performs an activity for a request raised on Foreign SPs.
- ✓ *time* – A series of scheduled time slots assigned to the resource to perform an activity.

It may also be extended to store additional information such as battery life, bandwidths, energy levels etc. of each resource.

3.2.4 Resource Allocation Manager (RAM)

The RAM **is** the main logic of selecting the sensor and actuator resources required for completing the task, owned by any SPs in the market. RAM initiates the Search Request by broadcasting it to the RAMs of all the other SPs (Foreign SPs) in the market. Each RAM executes the RS algorithm and addresses the problems specific to resource selection. It selects the best resource or combination of resources and passes their details to the scheduler.

3.2.5 Scheduler

The scheduler keeps track of the scheduled activities as per the request and triggers them by sending the request and resource information to the Execution Manager at its scheduled time.

3.2.6 Execution Manager (EM)

The EM establishes the connection among the selected resources obtained from RAM, by using the address information of the resources. This serves as a control manager for the tasks to be performed according to the execution plan.

3.2.7 SANET controllers

The Controllers share the important data in the system. The controller of each SANET sends the changes in the current status of its sensor and actuator resources to RM in order to update the local registry.

3.3 Flow of communication amongst the components in SATS:

The communication amongst the distributed and heterogeneous SPs is implemented through a common channel for homogenization and orchestration of messages, the Enterprise Service Bus, interconnected over the internet. The flow of the communication amongst all the components is carried out as follows:

Step-1: Collect user inputs.

Step-2: Divide into sub-tasks and get execution plan.

Step-3: Broadcast the Search Request.

Step-4: Execute RS algorithm.

Step-5: Update the local registry.

Step-6: Establish connection among selected resources.

The detailed explanation of the control flow steps in the SATS architecture is described next.

3.3.1 User Inputs

The SR gives the necessary inputs required to handle the request through an interface provided by the SP using an Internet connection from anywhere in the world. The SP with which the request was raised is termed as the Local SP, and all the other SPs in the market are classified as Foreign SPs, for that particular request. There could be any number of such requests at a particular instance. Each user request is given a unique requestID to differentiate it from other requests, and the following information is collected from the users:

- ✓ *Event:* The category of the activity to be performed (e.g. maintain temperature).
- ✓ *Condition*: The conditions on which the activity is to be performed (e.g. if temperature < 100F).
- ✓ *Action*: The actions to be performed by the SANET resources (e.g. setTemperature).
- ✓ *Duration*: The duration of the SANET resources to be active performing the activity (e.g. 09:00-13:00).
- ✓ *Location*: The exact locations of the area of event and area of action (e.g. LOC1430).
- ✓ *Time*: The starting time of the activity. Instantaneous or scheduled time (e.g. Scheduled).

3.3.2 Sub-tasks and Execution plan

All the inputs from the user are sent to the Interpreter Manager of the Local SP. The IM divides the request into sub-tasks if it can be split into independent activities which can be executed in parallel, and the flow of execution of tasks in performing an activity requested by the user is attached to the request.

3.3.3 Broadcasting the Search Request

The resulting requests from the IM are sent to the Resource Allocation Manager of the Local SP. For each request, it broadcasts a Search Request to the RAMs of Foreign SPs, in order to get the available resources from all the SPs in the market.

3.3.4 RS algorithm

Each RAM applies the RS algorithm on the current data present in its local registry. The computations for each request in the RS algorithm are handled by the Master and Slave nodes in the cluster of each SP. The Master node manages the assignment of resource data entries (records) from the local registry to multiple slave nodes. Slave nodes perform map, reduce, and combine tasks and also handle data motion among these phases. Each Service Provider has a dedicated Master, which acts like a parent node to all its child nodes (slaves). The number of slave nodes is chosen by the Service Providers considering the number of resources to process, and the selection time of the final resources.

An SP can have multiple such computational nodes and they can run both independently, in parallel. The algorithm takes the request information as input, and generates the number of resources required as the (*location*, *resourceType*) pairs. Master node initiates the processing of the request by reading the information in the local registry, and allocating a set of resource information in the registry to each slave node. Once the process is initiated, all the slave nodes run in parallel. Each slave node filters the resource information record for which the *availabilityFlag* and *activeFlag* are on. The physical range of the locations of requested resources, within the distance of radius 'k' units, is calculated in the form of (*location, resourceType*) pairs. These calculated new (*location, resourceType*) pairs within the radius are added to the requested (*location, resourceType*) pair, if the resource is within its proximity range of radius k. The scheduled time slots of the resulting resource are checked with the time of the execution of the request. If the resource is not busy during the time slot requested by the user, then the resource information is allowed to go through the next step of the algorithm. The Map and Reduce functions are then applied on the slave nodes as follows:

- Map: Parses each resource details in the set of records allocated by the Master node, and generates a list of intermediate <*resourceId, cost,* (*location, resourceType*)> set for each (*location, resourceType*) pair. The cost is calculated and added to the list. The usage charge is not applied on the resources if the search request is raised by its local SP. For foreign SPs, cost is applied by calculating the charges for the usage time of the resources.

  - ✓ Input: (*location, resourceType*)
  - ✓ Output: <*resourceId, cost,* (*location, resourceType*)>

- Reducer: Merges all the intermediate *<resourceId, cost, (location, resourceType)>* lists associated with the same intermediate *(location, resourceType)* pair. Accepts all the pairs generated by the Map and emits a count and sequence of *<list(resourceId, cost), (location, resourceType)>* for each *(location, resourceType)* pair.

✓ Input: Many <*resourceId, cost, (location, resourceType)*>

✓ Output: One <*list(resourceId, cost), (location, resourceType)*>

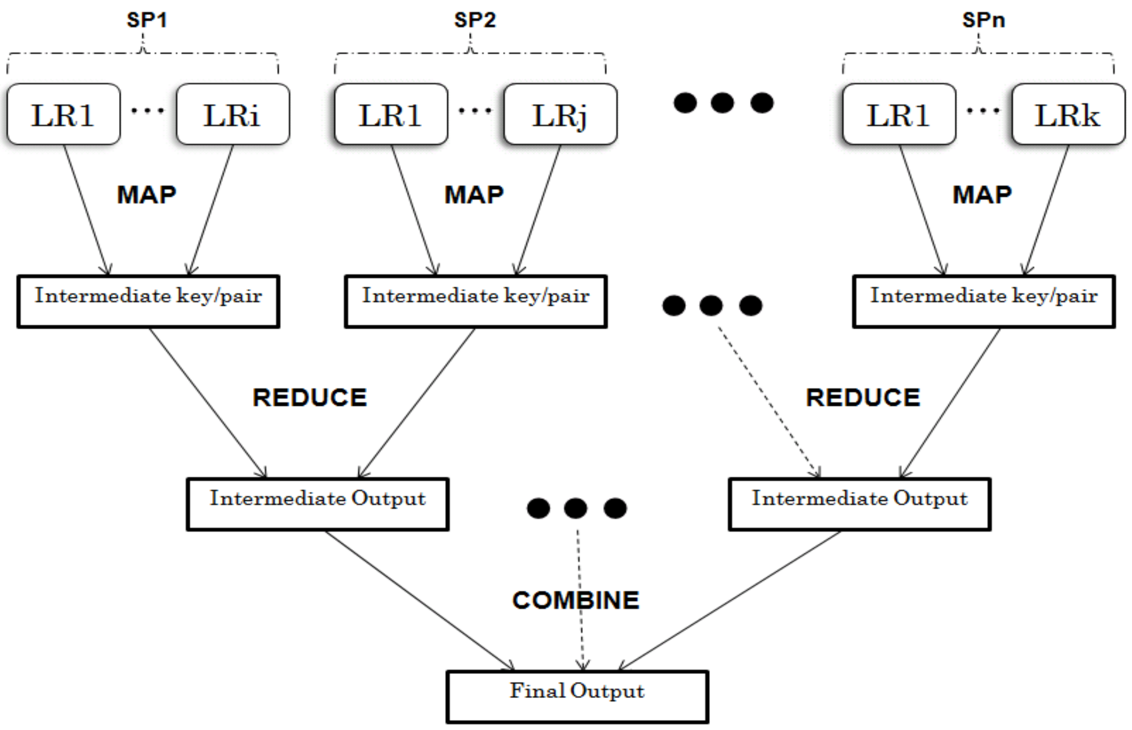

Figure 3: MapReduce framework used in SATS architecture

- Combiner: Same as Reduce. Accepts all the pairs generated by each Reduce function of all the Service Providers, and emits a count and sequence of <*list(resourceId, cost), (location, resourceType)*> for each *(location, resourceType)* pair.

✓ Input: Many <*list(resourceId, cost), (location, resourceType)*>

✓ Output: One <*list(resourceId, cost), (location, resourceType)*>

The resultant list has the information about all the resources that are idle in the market and capable of performing the task. Each list of the corresponding requested *(location, resourceType)* pair is ranked with the cost parameter obtained from the Map Reduce function. Considering *location* as k1, *resourceType* as v1, *resourceId* as k2, and *cost* as v2, the Map Reduce and Combine functions are represented as

$$\text{Map (k1, v1)} \rightarrow \text{(k2, v2)}$$

$$\text{Reduce (k2,v2)} \rightarrow \text{list(k2,v2)}$$

$$\text{Combine (list(k2, v2))} \rightarrow \text{list(list(k2,v2))}$$

The ranking parameter can be easily extended to consider other factors such as resource energy, SP’s preference list, etc. From the resultant list of ranked resources on the basis of cost, the first ranked resource is chosen for each requested *(location, resourceType)* pair. As there could be multiple requests on each resource at an instance, (i) the availability of the first ranked resource can change (ii) the first ranked resource can be scheduled to perform some other task with overlapping timings of the current requested time slot. These outcomes are possible upon the processing of other requests, while the selection of the current request is in process. Thus, the availability status and the scheduled timings of the first resource in the ranked list are checked with the entries in its local registry. . If the resource is scheduled to execute some other request or if it becomes unavailable, the algorithm selects the next ranked resource from the list and continues the process until the resource can be successfully mapped to the request. The resources selected are the best possible combination with minimal selection time.

3.3.5 Update the local registry

The local registries, to which the final selected resources belong to, are updated with their new status and scheduled time (if any).

3.3.6 Establish connection among selected resources

If the request is to be handled instantaneously, the request details are sent to the Execution manager. Otherwise, the scheduler handles the request as per its scheduled time by sending the request details to the Execution Manager.

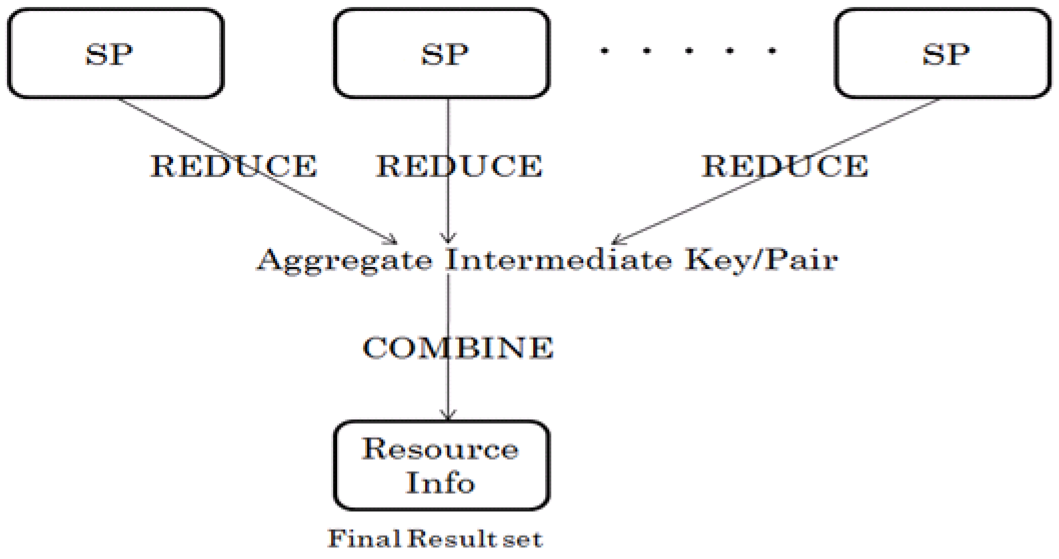

Figure 4: SP-level abstract view of Map Reduce implementation in SATS architecture

CHAPTER IV

SIMULATION

4.1 Objective of the simulation

The aim of the simulation is to validate the proposed RS algorithm to address problems specific to the Quality-of-Service parameters of the resource allocation in a large clustered SP environment. To achieve the goals with the minimal processing time for selection of resources, we implemented this algorithm on a varying number of large SPs in the market, with a varying number of computational nodes in each SP. For a request, the selection time for each combination is provided and its performance is understood from the graphs. For multiple overlapping resources in the requests, we provided the selection time for varying number of overlapping requests.

4.2 Development tools and programming languages

All the experiments were conducted on an AMD Opteron 2212 CPUs, with 4G of RAM (6G of virtual memory), running on CentOS 5.5. We implemented our algorithm in Java using Eclipse 3.5.1 IDE.

4.3 Assumptions

The architecture of the SATS is composed of various components involving the functionalities such as user web interface, web services, request interpretation, generating execution plan, message passing among different managers, etc. Therefore, we implemented RS algorithm in a java package with its inputs as the user request details, and output as the selected resources with backup resources list. We created a large resource information database (each file with a number of resources ranging from millions to billions, with database file sizes ranging from MBs to GBs) randomly in the local registry of each SP used in the experiment.

We assume each SP in the market with an equal number of resources, though each SP may own varying number of resources.

4.4 Experimental Design

We studied the effect of the RS algorithm on resource selection time and its Quality-Of-Service such as cost, timing constraints, service denial, failure recovery, and scheduling the tasks, with varying number of Service Providers and their nodes, and resources.

Our hypothesis is that the implementation of the RS algorithm in WSANs selects cost-effective resources that satisfy the request, with a minimal resource selection time, provides failure tolerance, and minimal service denial.

4.4.3 Network Units

- ✓ Network consisting of 5, 10, 15, 20, 25, 30 SPs.
- ✓ Each SPs consisting of 10, 20, 30, 40, 50 nodes.
- ✓ Each SPs consisting of 10M, 100M, 1B resources (M: Millions, B: Billions).

4.4.4 Independent Variables

- ✓ Request consisting of Resource Type, Location
- ✓ LocalSP – The SP with whom the request was raised.
- ✓ Proximity – The radius considered from the actual location of resource requested.
- ✓ Number of resources each SP has.
- ✓ Number of SPs, and their nodes.
- ✓ Time to use – Time slot that the resource should be available/dedicated to perform the requested task.

4.4.5 Control

The simulation is based on the Quality-of-Services provided by the Service Providers using the RS algorithm.

4.4.6 Replication

The experimental units have been simulated several times. An average value of all the trials is taken to plot the results.

4.4.7 Levels and Repeated Trials

LEVELS (NO. OF SPs IN THE NEWTORK, S)	5	10	15	20	25	30
FOR EACH LEVEL OF SP, NO. OF NODES IN THE SPs, N	10	20	30	40	50	60
REPEATED TRIALS	30	30	30	30	30	30

The above trails were performed for SPs with 10 Million and 100 Million resources each.

LEVELS (NO. OF SPs IN THE NEWTORK, S)	3					5				
FOR EACH LEVEL OF SP, NO. OF NODES IN THE SPs, N	10	20	30	40	50	100	150	200	250	300
REPEATED TRIALS	30	30	30	30	30	30	30	30	30	30

The above trails were performed for SPs with 1 Billion resources each.

NO. OF NODES IN THE SPs, N	10	20	30	40	50
NO. OF REQUESTS IN THE NETWORK	1000	2000	3000	4000	5000
REPEATED TRIALS	30	30	30	30	30

The above trails were performed in a network of 5 SPs, with a probability of overlapping requests as 1%.

4.4.8 Dependent Variables

- ✓ The best and the backup resource(s) are listed.
- ✓ Time (in msec) taken to select the best resources that satisfy the QoS parameters specific to resources. The calculated time is the difference between the system clock noted during the start and completion of the process/algorithm. The speed of execution of the process is also dependent on the system configuration on which it is executed, and several other common factors which affect the speed of a system. For consistency, several repeated trails were made and the average time is taken to plot the graphs.

4.4.9 Simulation Steps:

Let *s* be the number of SPs available in the market, *n* be the number of nodes available in each SP, *p* be the number of (*resourceType*,*location*) pairs requested by the user, and *k* be the number of resources, then the steps in the execution of the RS algorithm are as follows:

Step-1: Each SP_i cluster, where $1 \leq i \leq s$, consisting of one Master node '*M*' and *n* number of independent Slave nodes $\{N_1, N_2, N_3, \ldots N_n\}$, is configured.

Step-2: The resource type and the locations in the form $\{(resourceType_1, location_1), (resourceType_2, location_2), (resourceType_3, location_3), \ldots (resourceType_p, location_p)\}$, along with the physical radius '*P*' (optional) are collected from the user. These resource type and location pairs are stored in a list, *list_of_resources_requested*.

```
for (each input = (resourceType_i, location_i))
        list_of_resources_requested += (resourceType_i, location_i)
```

Step-3: The Master node '*M*' initiates the processing of the records in the registry $\{r_1, r_2, r_3, \ldots r_k\}$, where $r_i \in SP_i$.

Step-4: The Master node *M* allocates a set of records in the registry to each slave node allowing them to process independently.

```
for (each resource in local registry)
        random.Allocate(slave_node N_i)   where 1≤ i ≤ n
```

random.Allocate arbitrarily assigns each resource to a slave node belonging to its SP.

Step-5: The registry is filtered in parallel by the slave nodes, $\forall$ (r_k (*AvailbilityFlag*)=Y $\wedge$ r_k (*ActiveFlag*)=Y).

```
for (each resource $r_k$ assigned by its Master node)
    if (AvailbilityFlag$_k$ == Y && ActiveFlag$_k$ == Y)
        goto Step-6
    else
        resource_reject

        goto Step-4
```

Where *resource_reject* rejects the current resource to perform the requested task and continues the process by considering the next resource assigned by its Master node.

Step-6: The locations {$l_1, l_2, \ldots l_m$} of the $resourceType_p$ within the physical range '*P*' of the corresponding $location_p$ are calculated.

Step-7: The requested resource list is updated with the {l_m,$resourceType_p$} where l_m is in the physical range of $location_p$.

```
    if ( $l_i \leq coordinates(location_p) + P$ || $l_i \leq coordinates(location_p) - P$)
        list_of_resources_requested += (resourceType$_i$, $l_i$)
```

Step-8: Map function is applied on the resulting records such that $r_k(location)=location_p$, $r_k(resourceType)=resourceType_p$, and $l_m \in r_k(resourceType,location)$. Hence each slave node N_n generates *intermediate_list_map*, a list of intermediary *(resourceType,location)* key and *(resourceId, cost)* value pairs, resulting in a total of '*n*' lists; of *type* ($resourceType_p$, $location_p$).

```
        if ( $location_k == location_p$ && $resourceType_k == resourceType_p$)
            $intermediate\_list\_map += (resourceId_k, cost_k, (rescourceType_p, location_p))$
```

Step-9: These *n* lists are combined and categorized into *p* sub-lists; of type *(resourceType$_p$, location$_p$)* using the reduce function. (This gives the list of available resources with each Service Provider, at the locations specified by the user, and also within the physical range of the requested locations). The resulting combined list is denoted as *list_reduce*.

```
    for (each intermediate_list_map.element)
        $list\_reduce(resourceType_p,location_p) +=$
        $intermediate\_list\_map.element(resourceType_p,location_p)$
```

Step-10: Step-3 to Step-8 is repeated for each SP_i, where $1 \leq i \leq s$.

Step-11: The list obtained from each SP_i is combined to form a single list consisting of all the resources which can perform the requested tasks, using the Combine function. (This gives the list of available resources with all the Service Providers available in the market, at the locations specified by the user, and also within the proximity range of the requested locations). The resulting combined list is denoted by *list_combine*.

```
    for (each list_reduce.element)
        $list\_combine(resourceType_p,location_p) +=$
        $list\_reduce.element(resourceType_p,location_p)$
```

Step-12: The resulting list is ranked on the basis of cost parameter within the list.

```
for(each list_combine(resourceType$_p$,location$_p$))
    sort(list_combine(resourceType$_p$,location$_p$), cost)
```

Step-13: The highest ranked resource is selected from each sub-list. Its latest status and the scheduled time are checked. If the requested time overlaps with the scheduled time in the resource details in the local registry, the next ordered resource in the list is selected and this process continues until a resource is found without any overlapping time slots. The resource without any overlapping time slots is selected and the scheduled time slots are updated with the requested time slot.

```
for(each list_combine(resourceType$_p$,location$_p$))
    resource_selected=resource_list(resourceType$_p$,location$_p$).first ()
    while(resource_finalized_flag == false)
        if (requested_time_slot in resource_selected(scheduled_time_slots))
            resource_selected=
            resource_list(resourceType$_p$,location$_p$).next ()
        else
            resource_selected(scheduled_time_slots).update()
                =requested_time_slot
            resource_finalized_flag=true
```

Step-14: Repeat Step-13 for each resource requested, with the list of available corresponding resources in the market.

The output of the RS algorithm after step-14 is the best available resources for all the requested resources in the request, after considering all the QoS parameters.

The results of our experiments are shown below. The experiments are performed with a varied number of SPs, nodes, and resources available at each SP. The final graph is a probability of 1% of overlapping requests on a resource at an instance of time against varying number of nodes.

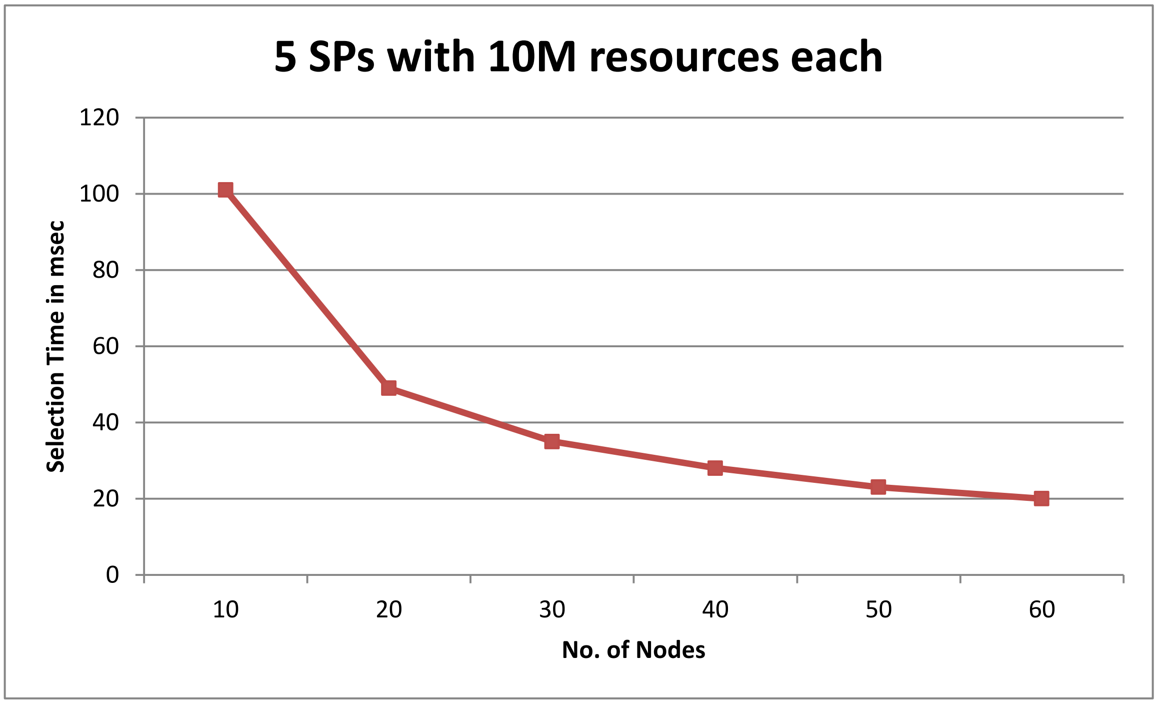

Figure 5: Selection time (in msec) vs. number of nodes in a market of 5 Service Providers, each with 10 million resources.

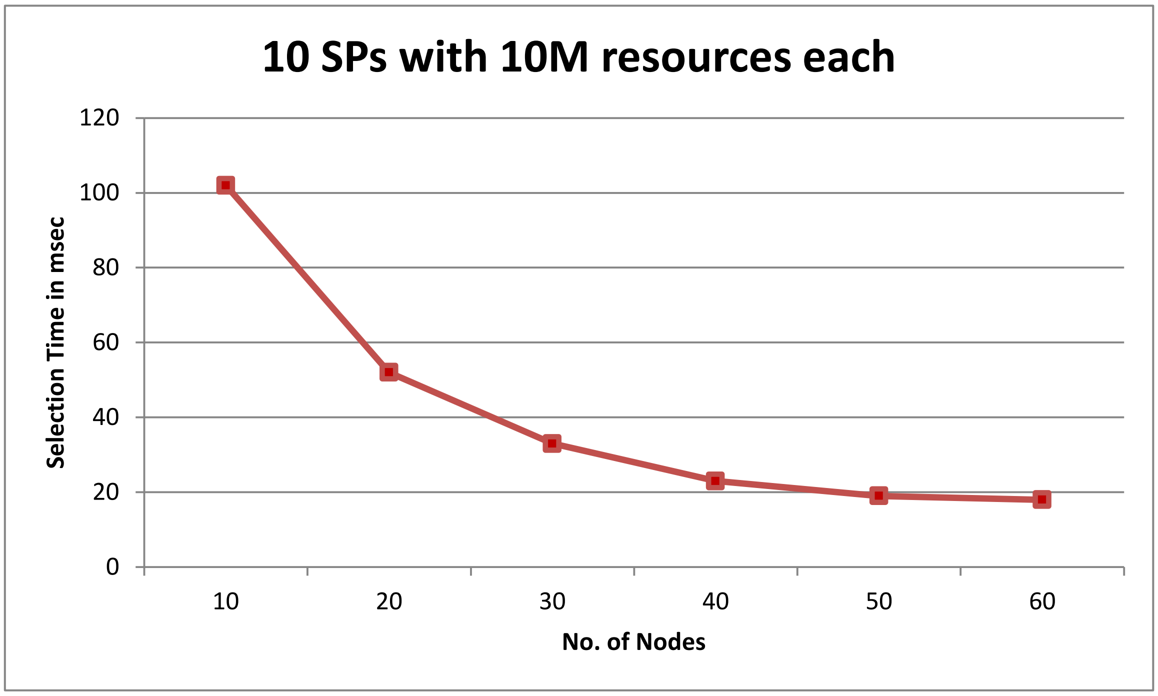

Figure 6: Selection time (in msec) vs. number of nodes in a market of 10 Service Providers, each with 10 million resources.

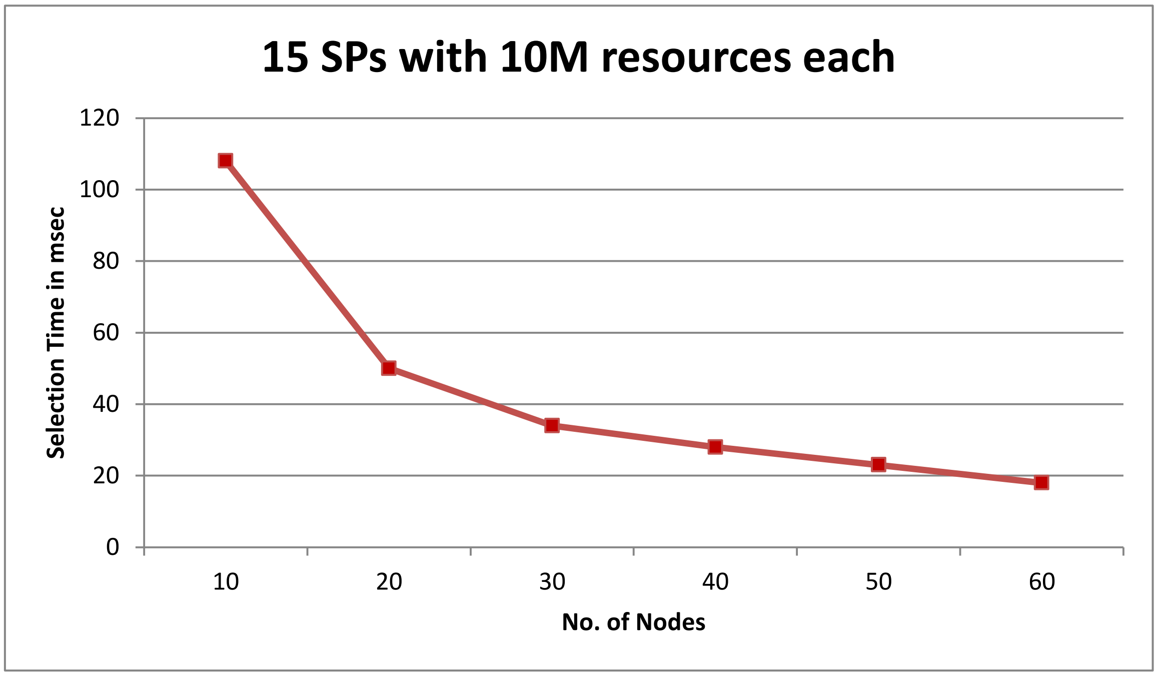

Figure 7: Selection time (in msec) vs. number of nodes in a market of 15 Service Providers, each with 10 million resources.

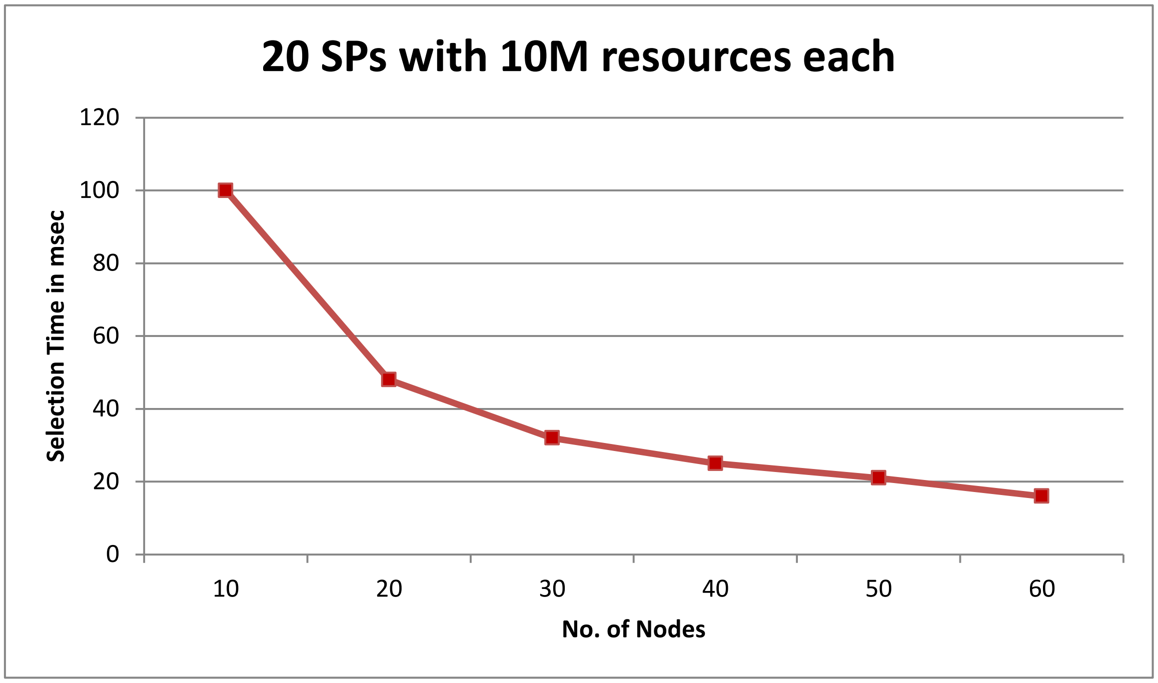

Figure 8: Selection time (in msec) vs. number of nodes in a market of 20 Service Providers, each with 10 million resources.

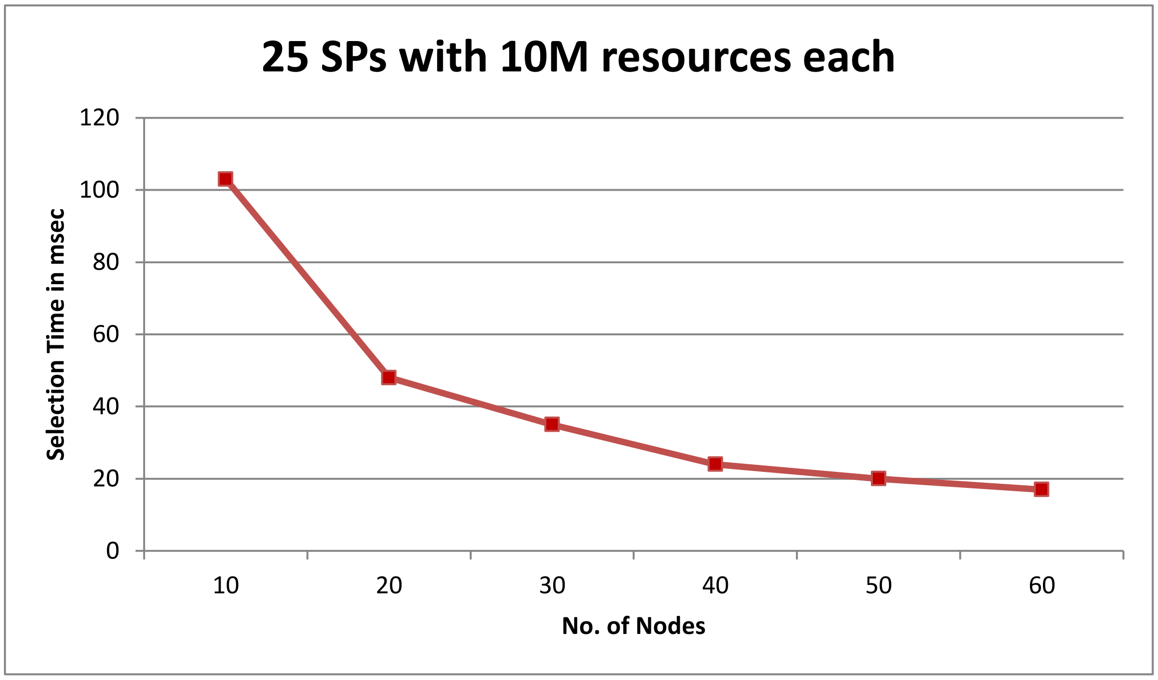

Figure 9: Selection time (in msec) vs. number of nodes in a market of 25 Service Providers, each with 10 million resources.

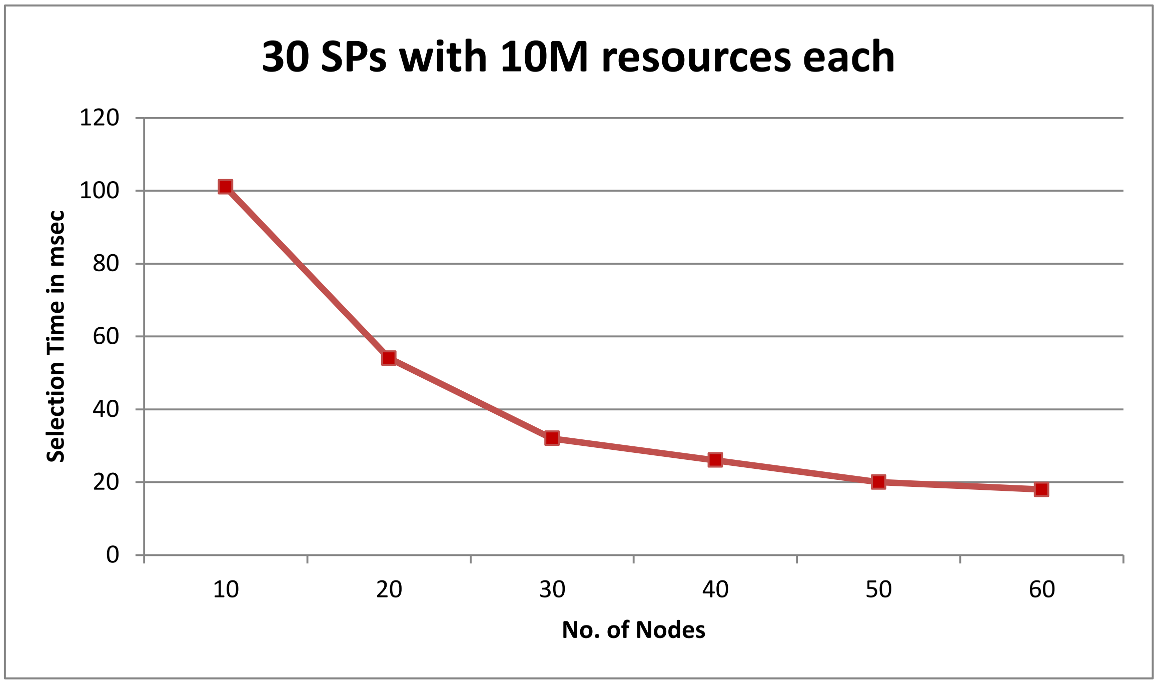

Figure 10: Selection time (in msec) vs. number of nodes in a market of 30 Service Providers, each with 10 million resources.

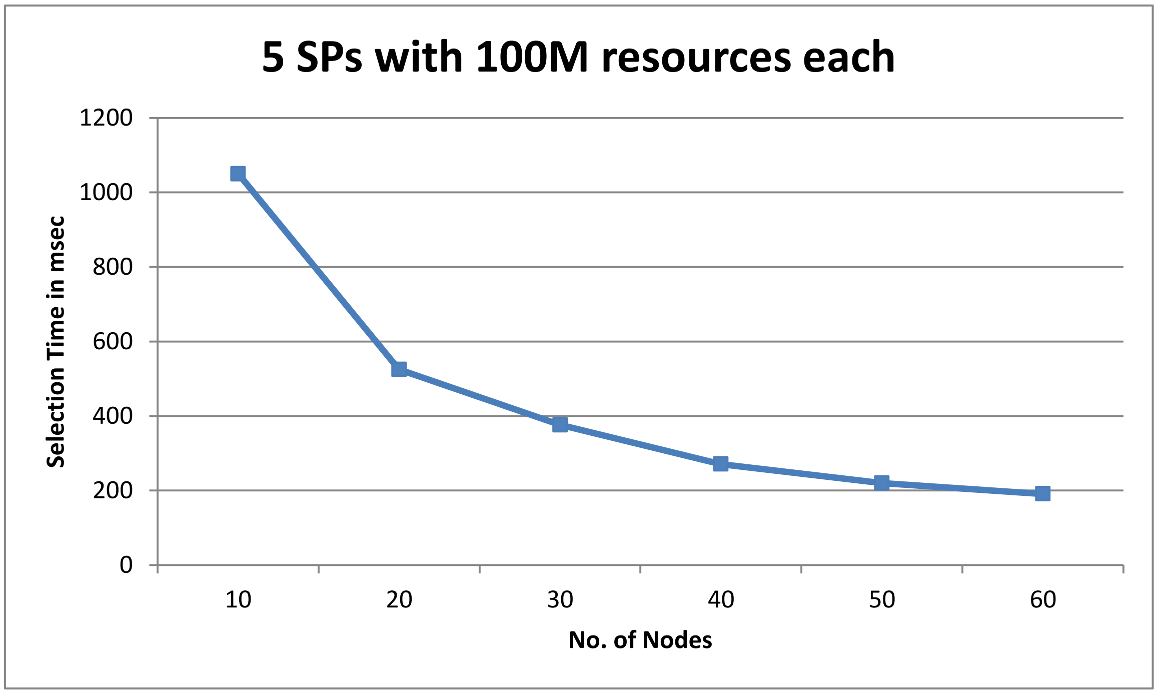

Figure 11: Selection time (in msec) vs. number of nodes in a market of 5 Service Providers, each with 100 million resources.

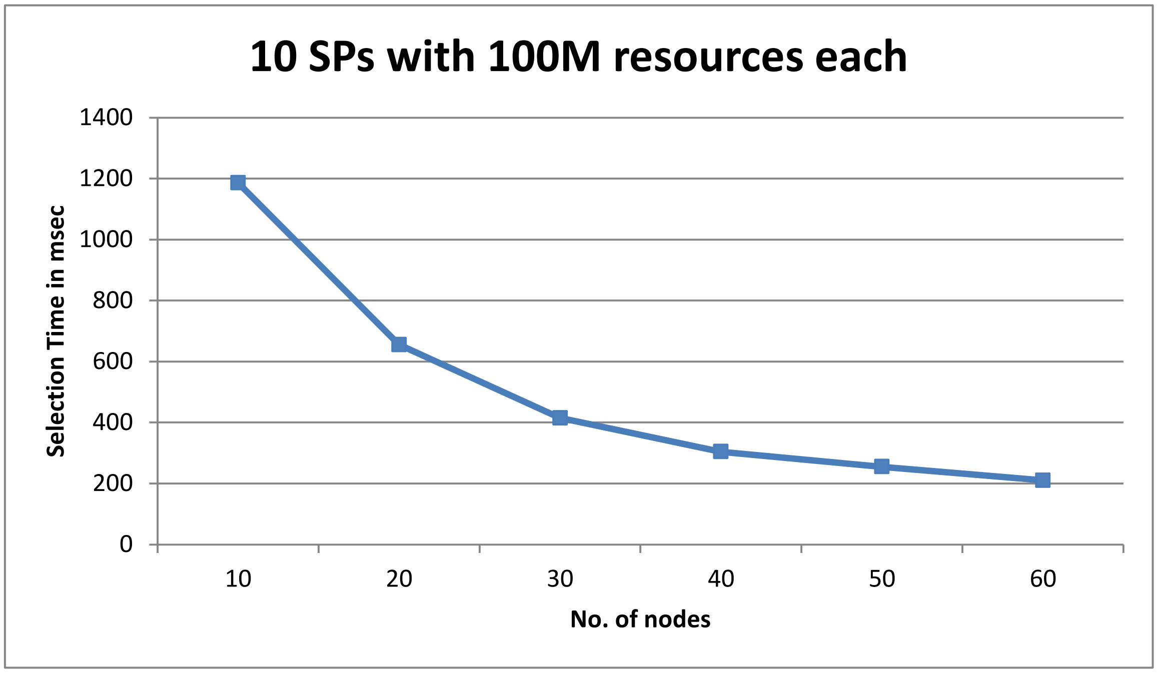

Figure 12: Selection time (in msec) vs. number of nodes in a market of 10 Service Providers, each with 100 million resources.

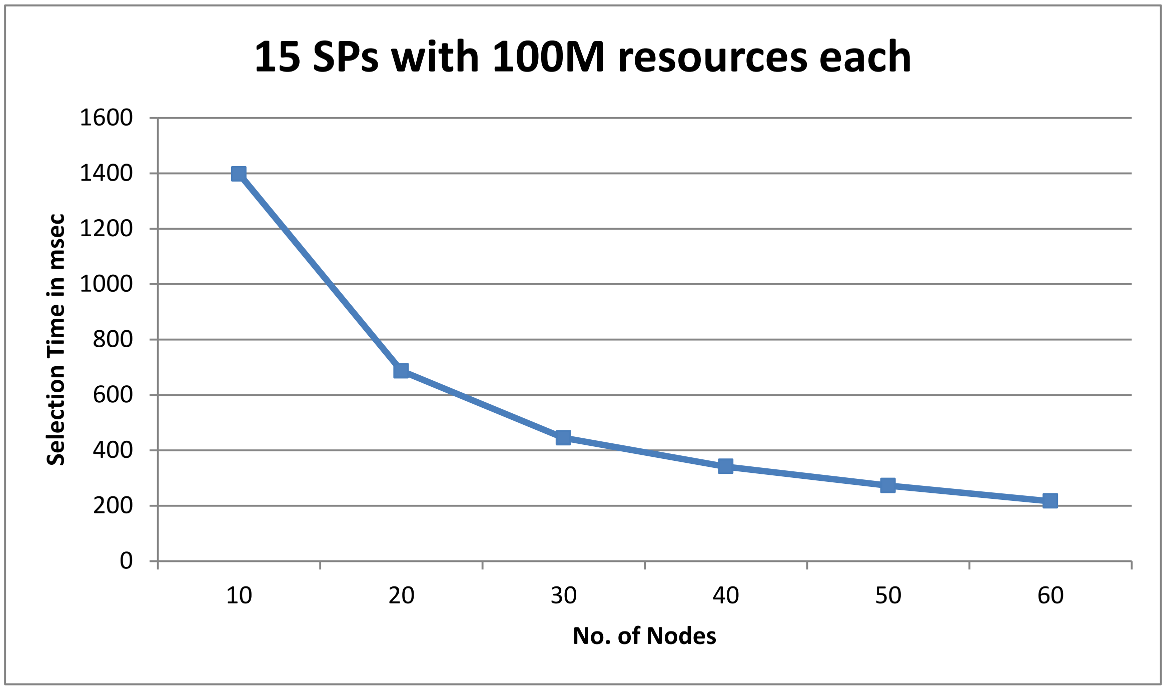

Figure 13: Selection time (in msec) vs. number of nodes in a market of 15 Service Providers, each with 100 million resources.

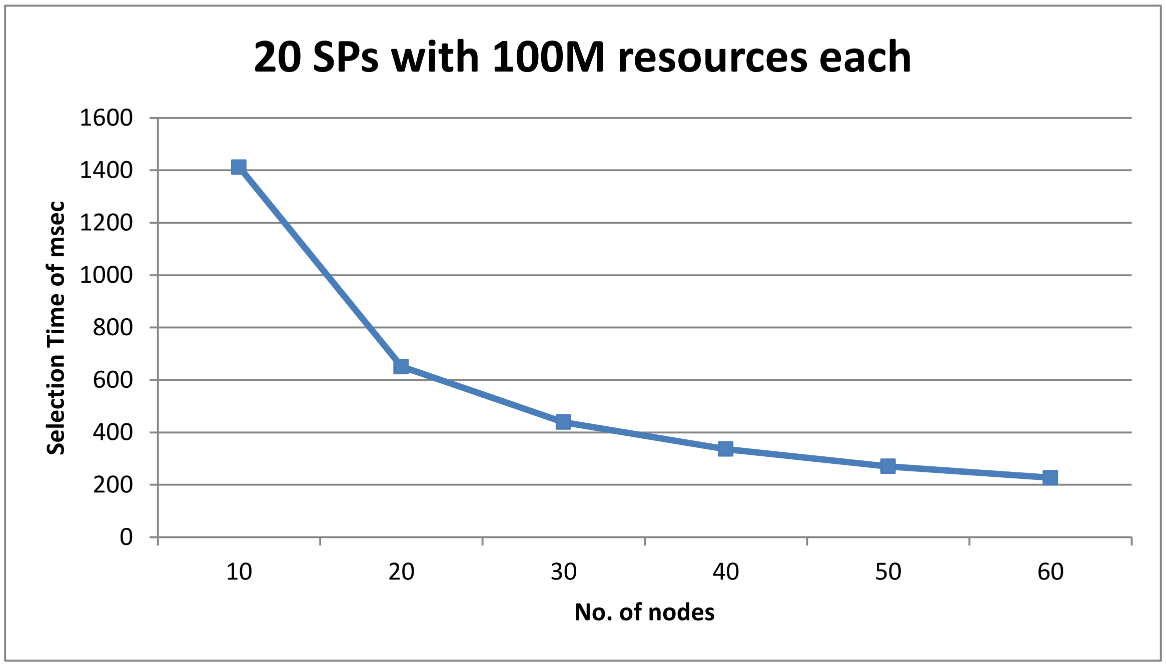

Figure 14: Selection time (in msec) vs. number of nodes in a market of 20 Service Providers, each with 100 million resources.

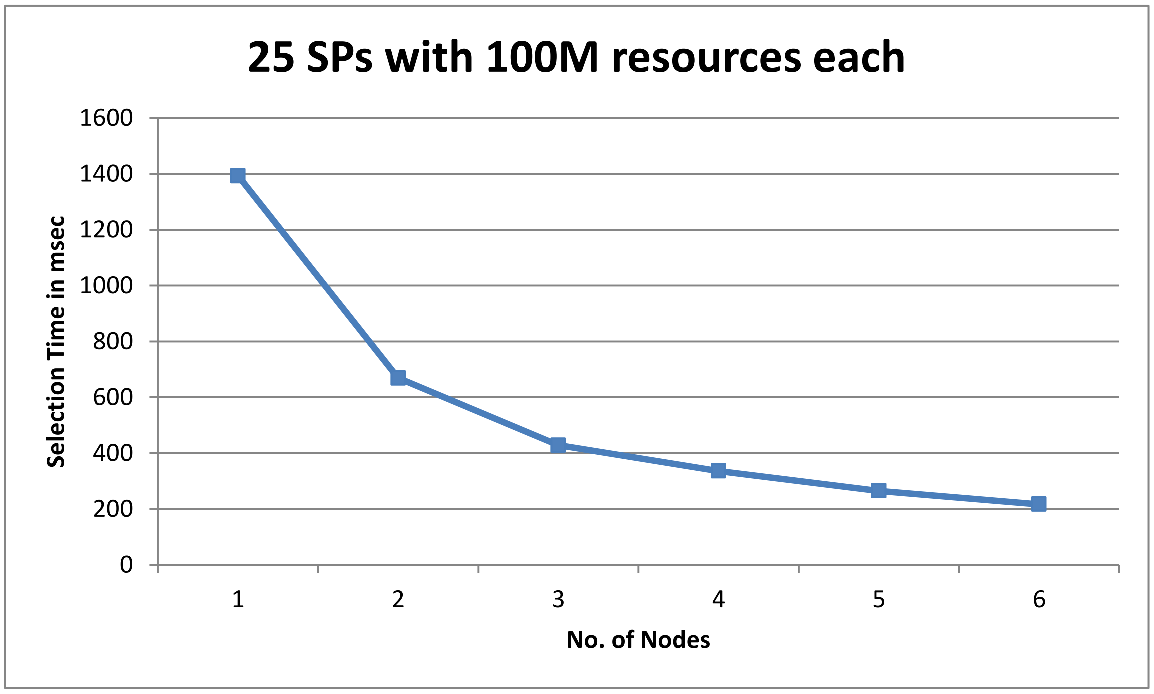

Figure 15 Selection time (in msec) vs. number of nodes in a market of 25 Service Providers, each with 100 million resources.

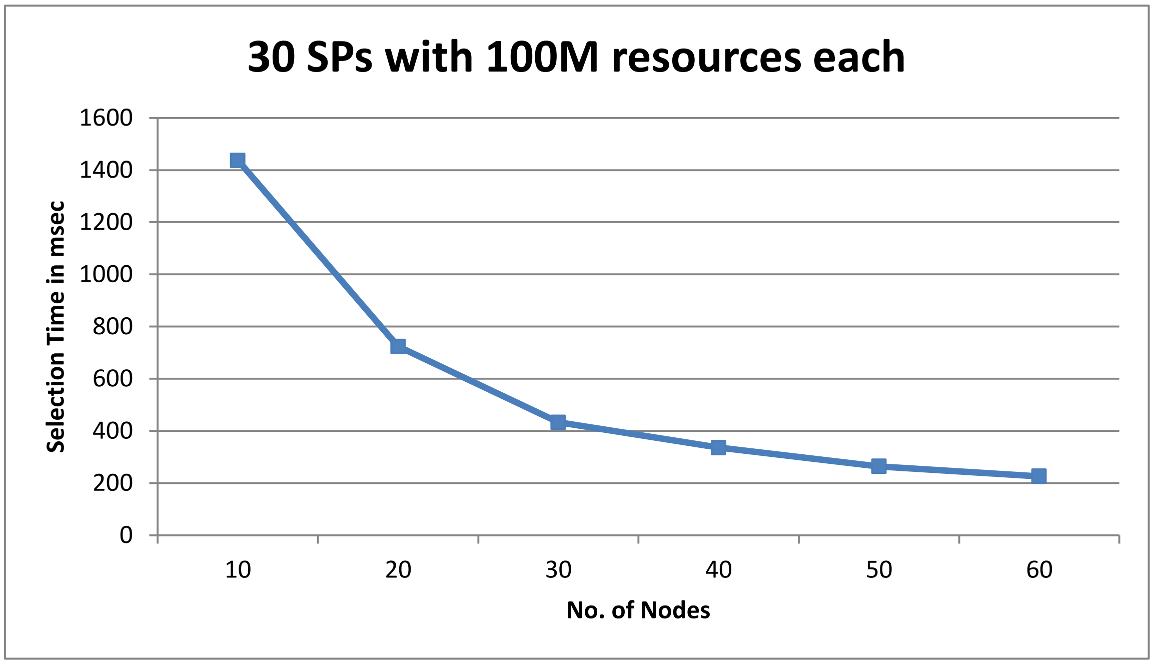

Figure 16 Selection time (in msec) vs. number of nodes in a market of 30 Service Providers, each with 100 million resources.

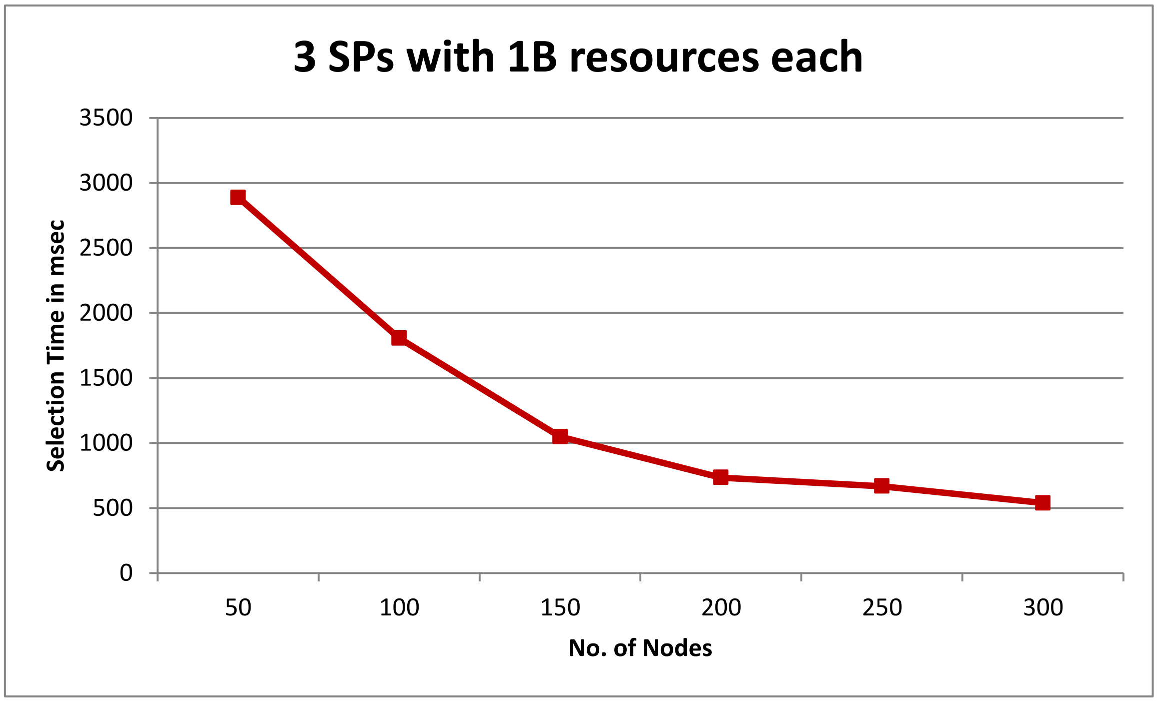

Figure 17 Selection time (in msec) vs. number of nodes in a market of 3 Service Providers, each with 1 billion resources.

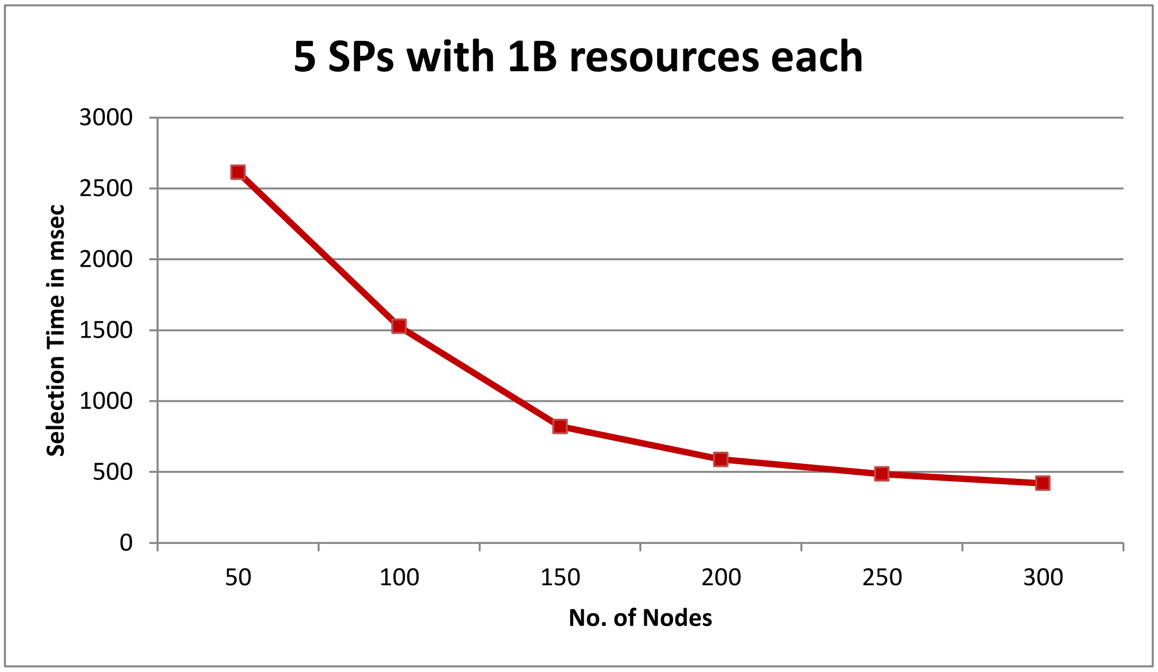

Figure 18: Selection time (in msec) vs. number of nodes in a market of 5 Service Providers, each with 1 billion resources.

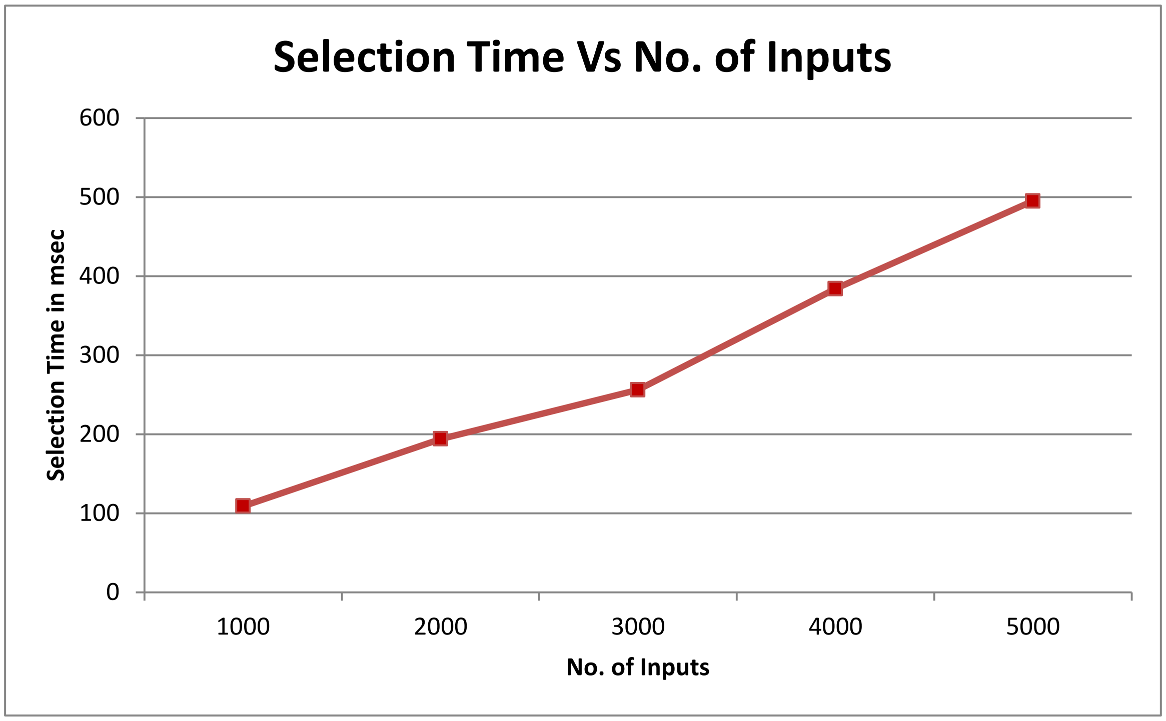

Figure 19: Increase in selection time (in msec) vs. increase in number of overlapping resource inputs.

4.5 Observations

From the graphs depicting the selection time for varying number of SPs, nodes, and number of resources, we observe that the selection time is affected with the change in the number of computational nodes, and number of resources. The selection time decreases significantly with the increase in the number of nodes on each SP, as parallel computation dominates aggregation of lists from each node. Moreover, the selection time varies negligibly with increase in the number of nodes after a certain limit, as the aggregation impacts the parallel computation.

The number of SPs does not impact the selection time much, as they execute in parallel. However, there is a processing overhead due to the aggregation of data from each SP.

It is also observed that the increase in overlapping resource requests increases the selection time due to the overhead of checking the latest availability status of each resource in the final list of ranked resources, sequentially. This process continues until it finds a resource, from the final list, which has not been mapped to another request.

CHAPTER V

CONCLUSIONS

With growing technology developments and increase in the usage of sensing and actuating resources in day-to-day life, there is a need for an efficient and robust architecture to provide these resources as services in an efficient and cost effective manner.

The goal of the SATS architecture is to provide inter-operability among SP instances in the market, with an ability to efficiently search and schedule the resources available in order to execute the sensing and actuating activities requested by the user. Some of problems to be addressed that are specific to resource selection include satisfying the QoS by considering possible service denial, efficient resource utilization, satisfying timing constraints, resource information hiding, cost constraints, failure tolerance, overlapping and location dependent resource requests**.** In addition to addressing these QoS related problems, the goal is to have a minimal selection time to map the resources to a request.

To support the above QoS, we proposed a RS algorithm based on a Map Reduce framework. We simulated an application implementing the RS algorithm on the current status of each resource belonging each available SP. Simulations to measure selection time while satisfying the QoS problems, the change in the selection time with varying number of resources, overlapping resource usage requests, etc. were conducted**.** The results demonstrate the effect of the number of nodes and resources in each SP on the selection time of the cost effective resources available in the market, to perform the requested tasks. The proposed architecture will provide the optimal selection time with increasing QoS.

CHAPTER VI

FUTURE WORK

With the rapid increase in the usage of Sensors and Actuators to perform day-to-day activities, future work should investigate the implementation of the SATS architecture with real time Sensors and Actuators, and determine its performance. The implementation of the Interpreter, which divides a request into sub-tasks and generates the execution plan, will be be addressed. The execution of the tasks as per the scheduled time with the selected resources should be implemented using a real-time scheduling algorithm that works at the resource level.

REFERENCES

1. T. Melodia, D. Pompili, V. C. Gungor, A. F. Akyildiz, “Communication and Coordination in Wireless Sensors and Actors Networks”, *IEEE Transactions on Mobile Computing*, Vol. 6, No. 10, pp. 1116-1129, October 2007.

2. F. Xia, Y.C. Tian, Y.J. Li, Y.X. Sun, “Wireless Sensor/Actuator Network Design for Mobile Control Applications”, *Sensors* , Vol. 7, No. 10, pp. 2157-2173. 2007

3. T. Melodia, D. Pompili, I. F. Akyildiz, “Handling Mobility in Wireless Sensor and Actor Networks”, *IEEE Transactions on Mobile Computing*, Vol. 9, No. 2, pp. 160-173, Feb 2010.

4. A. Rezgui, M. Eltoweissy, “Service-oriented sensor-actuator networks: Promises, challenges, and the road ahead”, *Computer Communications*, Vol. 30, pp. 2627-2648, 2007.

5. J.W. Branch, L. Chen, and B.K. Szymanski, “A Middleware Framework for Market-Based Actuator Coordination in Sensor and Actuator Networks”, *ACM International Conference on Pervasive Services, Sorrento, Italy*, *ACM Press*, pp. 101-110, July 6-10, 2008.

6. F. Xia, “QoS Challenges and Opportunities in Wireless Sensor/Actuator Networks”, *Sensors*, Vol. 8, pp. 1099-1110, 2008.

7. J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *6th Symposium on Operating System Design and Implementation, San Francisco, CA*, Dec. 2004.

8. *Wireless sensor network*, http://en.wikipedia.org/wiki/Wireless_sensor_network, (last accessed Jan 22, 2011)

9. A. Koubaa, M. Alves, “A Two Tiered Architecture for Real-Time Communications in Large-Scale Wireless Sensor Networks: Research Challenges”, *IRISA Research Report*, number PI-1723, pp. 33-36, July 2005.

10. I. A. Ismail, I. F. Akyildiz, I. H. Kasimoglu, “Wireless sensor and actor networks: Research challenges”, *Ad Hoc Networks*, Vol.2, pp. 351–367, 2004.

11. F. Dressler, I. Dietric, R. German, B. Kruger, "A Rule-based System for Programming Self-Organized Sensor and Actor Networks", *Elsevier Computer Networks*, Vol. 53, pp. 1737-1750, 2009.

12. S. Craciunas, A. Haas, C. Kirsch, H. Payer, H. R¨ock, A. Rottmann, A. Sokolova, R. Trummer, J. Love, and R. Sengupta, "InformationAcquisition-as-a-service for Cyber-physical Cloud Computing," *2nd USENIX conference on Hot topics in cloud computing*, pp. 14, 2010.

13. M. Krüger, R. Karnapke, J. Nolte, "Controlling Sensors and Actuators Collectively Using the COCOS-Framework", *ACM Conference on SANET, Montréal, Québec, Canada*, pp. 53-54, 2007.

14. S. Sastry, S. S. Iyenger, " Real-Time Sensor–Actuator Networks", *International Journal of Distributed Sensor Networks*, Vol. 1, pp. 17–34, 2005.

15. *The Future of Cloud Computing*, http://pewresearch.org/pubs/1623/future-cloud-computing-technology-experts (last accessed Feb 2, 2011).

16. D. Martíne, F. Blanes, J. Simo, A. Crespo, "Wireless Sensor and Actuator Networks: Characterization and Case Study for Confined Spaces Healthcare Applications", *International Multi-conference on Computer Science and Information Technology*, pp. 687 – 693, 2008.

17. *Large Scale Data Analysis with Map/Reduce*, http://www.slideshare.net/marin_dimitrov/large-scale-data-analysis-with-mapreduce-part-i (Last accessed Feb 6, 2011).

18. S.F. Li, "Wireless sensor actuator network for light monitoring and control application", *3rd IEEE Consumer Communications and Networking Conference*, Vol. 2, pp. 974- 978, 2006.

19. D. I. Curiac, C. Volosencu, "Urban Traffic Control System Architecture Based on Wireless Sensor-Actuator Networks", *2nd International Conference on Manufacturing Engineering, Quality and Production Systems*, pp. 259-263, 2010.

20. T. Wark, C. Crossman, W. Hu, Y. Guo, P. Valencia, P. Sikka, P.I. Corke, C. Lee, J. Henshall, K. Prayaga, J. O'Grady, M. Reed, A. Fisher,"The design and evaluation of a mobile sensor/actuator network for autonomous animal control", *International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 206-215, 2007.

21. *Cloudera MapReduce Algorithms*, http://www.scribd.com/doc/13305093/Hadoop-Training-5-MapReduce-Algorithm (Last accessed Feb 10, 2011).

22. *Understanding Service Oriented Architecture*, http://msdn.microsoft.com/en-us/library/aa480021.aspx (Last Accessed Jan 20, 2011).

23. T. Rajendran, P. Balasubramanie, "Analysis on the Study of QoS-Aware Web Services Discovery", *Journal Of Computing*, Vol. 1, No. 1, pp. 119-130, Dec 2009.

VITA

Thrishukanth Dasari

Candidate for the Degree of

Master of Science

Thesis: SENSING AND ACTUATING TASKS AS SERVICES AND ITS QUALITY OF SERVICES IN LARGE CLUSTERED ENVIRONMENTS

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in July, 2011.

Received the Bachelor of Engineering degree in Computer Science at Osmania University, Hyderabad, AP, India in 2007.

Experience:

Graduate Technical Linux Lab Assistant at Physical Sciences, Oklahoma State University, Stillwater, OK — July 2010- May 2011

Research Assistant under Dr. Johnson P. Thomas, Oklahoma State University, Stillwater, OK — Jan 2010- June 2010

Software Engineer at Mahindra Satyam Computer Services Limited, Hyderabad, India — June 2007- July 2009

Intern at Tata Consultancy Services, Hyderabad, India — Dec 2006- June 2007

Name: Thrishukanth Dasari Date of Degree: July, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: SENSING AND ACTUATING TASKS AS SERVICES AND ITS QUALITY OF SERVICES IN LARGE CLUSTERED ENVIRONMENTS

Pages in Study: 49 Candidate for the Degree of Master of Science

Major Field: Computer Science

With the proliferation of sensors and actuators in today's world, we envisage the world as an inter-connected network of millions of Sensing and Actuating resources performing multiple tasks in everyday life. These distributed resources are capable of performing tasks that monitor and/or affect the parameters of the physical and environmental entities. To perform a task, the user might require a single or group of sensors and/or actuators, which are offered by multiple Service Providers in the market.

The ability to trigger these tasks without the user having to determine the owner of the service, schedule tasks by searching and determining the availability of resources, in a location-independent manner, is provided by enabling the Sensor and Actuator resources as services. We propose an architecture called SATS (Sensing and Actuating Tasks as Service) that provides the ability to trigger sensing and actuating tasks over the Internet by selecting the best combination from the available resources, including the resources owned by other Service Providers. Selection of the best possible resources amongst the available resources is a challenge as many problems related to QoS have to be addressed. We propose a solution based on the Map-Reduce framework and develop the RS (Resource Selection) algorithm to address the problem of resource selection, in a network of service providers provide sensing and actuator services that are composed of large numbers of sensors and actuators.

ADVISER'S APPROVAL: Dr. Johnson P. Thomas