PIRR: a Methodology for Distributed Network Management in Mobile Networks

FILIPPO NERI
University of Piemonte Orientale
Department of Science
via Bellini 25/g, 13900 Alessandria
ITALY
filipponeri@yahoo.com

Abstract: The current centralized Network Management approach in mobile networks may lead to problems with being able to scale up when new customer services and network nodes will be deployed in the network. The next generation mobile networks proposed by 3rd Generation Partnership Project (3GPP) envision a flat network architecture comprising some thousands of network nodes, each of which will need to be managed, in a timely and efficient fashion. This consideration has prompted a research activity to find alternative application architectures that could overcome the limits of the current centralized approaches. section approach to deal with the increasing network size and complexity is to move from a centralized Network Management system to a more distributed or decentralized approach where each Network Management application consists of a controller part and a set of distributed parts running on the individual network elements. This approach however raises questions of how to measure and estimate the resource requirements when planning to distribute the Network Management applications in the mobile networks before starting to produce commercial systems. In this paper, we describe the PIRR methodology we have developed to measure resource requirements for distributed applications in mobile networks and the experimental findings of its application to new Network Management applications.

Key–Words: Software Agents, JADE, Telecom network management system, distributed simulation systems, 3G cellular system.

1 Introduction

The basic research problem we want to study in our work is how to measure and estimate the resource requirements and performances of deploying distributed applications into the network management (NM) system of a mobile telecommunications network[1].

Moreover we would like to use application specific performances, together with information about the network configuration,

1. to compare alternative versions of an application at the design time to decide which one to further engineer, or

2. to decide at runtime if or not to run a distributed application by dynamically estimating its resource requirements.

In order to provide some background knowledge about mobile networks and their network management systems, we will start to introduce a simplified model of a mobile network. We present in the next section a more detailed description of the main components of a 2G/3G mobile network and how they cooperate to provide user services, such as voice calls and data transfer. From the point of view of our research, a telecom mobile network can be viewed as a set of several types of network elements responsible for routing the voice or data traffic across the network. In particular for this research, we focus our attention to one Network Element: the OSS (Operation Support System) node and on the set of Network Management applications running on it. The OSS system assists the network operator (for example, a company such as Vodafone, TMobile, etc.) in configuring and monitoring the operation status of each network element and of the network as a whole. Among other activities, the OSS constantly updates a centralised database recording the network configuration, by collecting data from the network elements, and by propagating changes across the network when a new configuration is defined. State-of-the-art commercial telecom networks adopt a centralized solution which resides and runs

[1]This work was partially funded through a Marie Curie fellowship grant and done at Ericsson LMI Software Campus, Athlone, Co. Westmeath, Ireland

on the OSS node. As can be easily understood with the centralized approach, the OSS node become the focus of high computational activity and of intense message traffic which in turn require the provision of expensive hardware and large amounts of bandwith. As an example, status data from the Network Elements is copied to the OSS in order to be manipulated and any new configuration data is pushed back to the Network Elements affected by the configuration change. This way of dealing with Network Management tasks results in scalability issues, in term of number of Network Elements that can be controlled by one OSS, and potentially problems also with data consistency, as Network Element data may have to be duplicated across the network. Therefore the centralized approach can result in the OSS having to operate with an outdated picture of the network, which may then result either in the propagation of configuration errors when, for example, new Network Elements have to be included into the network, or it may result in the inability of the network manager to detect when a customer service, (such as video call), is not working properly because of a technical fault somewhere in the network. A potential solution approach to the outlined problems could be to decentralise or distribute more of the Network Management functionality directly to the network elements. This would have the advantage of dealing with configuration changes at, or close to, the data source, thus reducing issues with data consistency, while at the same time freeing computational resources on the OSS node, and reducing bandwith requirements to/from the OSS node, and allow for a more scalable network architecture supporting a greater number of Network Elements being managed from one OSS instance.

2 Related works

In [8], the authors show that predictive algorithms can be successfully employed in the estimation of performance variables and the prediction of critical events in system management tasks.

In [2], the authors describe an off-line modeling tool able to predict the impact of changes to a network's topology, configuration, traffic, and technology.

In [6], Mobile Ad hoc NETworks (MANETs) are studied. MANETs are dynamic networks populated by mobile stations.

Ns-2 [7] is the de facto standard for network simulation. We are considering to use the ns-2 simulator [7] for some advanced stage work on mobile network modeling.

GloMoSim [10] from UCLA is the second most popular wireless network simulator. Lack of documentation makes difficult to adopt and exploit the simulator.

The OPtimized Network Engineering Tools, OPNET, [3] is a network simulator proposed by MIT in 1986. It is a well-established and professional commercial suite for network simulation.

Researchers at IBM [9, 5] report two approaches to measure and predict performance behaviour under growing workload for a centralized system.

3 Distributing NM to deal with new services and more nodes in telecom wireless networks

As mentioned earlier, the current centralized NM system may run into problems when attempting to scale up when new customer services and network nodes will be deployed across a mobile network. This consideration has prompted a research activity to find alternative architectures that could overcome the limits of the current centralized approaches.

A straightforward idea to deal with increasing network size and complexity is to move from a centralized NM system to a distributed/decentralized one where each NM application previously residing in the OSS node is broken into a controller part running on the OSS node and a distributed part running on each affected network element. While such suggestion is easy to promote, the difficulty lies in proving that the suggestion can actually work and that actually improve the NM operations while at the same time ensuring that the performance of the network, in carrying traffic, is not adversely affected. In the following section we will review the tools available for simulating or modeling part of the telecom wireless network and their limitations.

4 Selected mobile network simulators

In our research, we would like to be able run code both on NEs and on the OSS node, and we then would like to use an experimental mobile network to verify the effect produced by our code. This is easier said that done as our experience proved.

Accessing an experimental mobile network is not an easily achievable task as priority is obviously given to the commercial production projects. Therefore we decided to run our experiments on the JADE platform [1, 4] and then to move the code to the experimental mobile network when the procedures and the code for the experiments are tuned.

The JADE (Java Agent DEvelopment Framework) is a java based middleware that simplifies the implementation of multi-agent systems. We can use it as we have based the development of our distributed NM applications on Object Oriented programming and the Software Agent paradigm. It is therefore straightforward to prototype alternative versions of an application in JADE and simulate the distributed NM system of a mobile network by using a network of cooperating agents where each agent implements one NE.

5 Identifying relevant Performance measures

In order to measure the impact of deploying distributed applications in a mobile network, a set of measures has to be defined which should allow to identify requirement for relevant limited resources. For the scope of this research, we have defined the following measures as relevant, but the list can be extended on a case by case need:

1. CPU usage - average CPU usage by the decentralized application over time
2. Memory utilization - average memory usage over time
3. Storage - maximum amount of hard disk space needed
4. Bandwith - average bandwith utilization.

In particular, we are interested in monitoring variations in those measures under different load conditions and ideally we would like to be able to estimate how an application will perform under unseen load conditions (number of nodes in the network).

6 The PIRR estimation methodology

Given our research problem, we would like to develop a performance and resource estimation methodology which could predict the impact of deploying a distributed application, NewApp, into the mobile network. Because of the complexity of the problem we are dealing with, and the limits of the available simulators, we decided to base our estimation methodology on a combination of empirical and analytical practices.

We will call our methodology PIRR (pronounced as the word 'peer') from Performance Indexes and Resource Requirements estimation, and it consists of a two step approach.

In the **first phase**, we will run NewApp in our simulation environment and collect as much relevant performance data as possible by using profiling tools. Unfortunately not all the information we are interested in is ready available from a profiling tool.

In the **second phase**, we will analyze the code of NewApp in order to understand what part of the algorithm affects the application's performances. On the base of this analysis, we will develop a mathematical model of NewAppl's performances that should be able to explain the set of observed performance data and should allow to estimate the PIRR values when NewApp runs in different network loads.

We will show on an example how the PIRR methodology can be applied.

7 The PIRR methodology applied to the OK-PING service

In order to show the PIRR methodology at work, we selected two versions of the simplified OK-PING application.

In the rest of the section, we will described the two versions of the OK-PING application and we will show how we were able to collect and estimate PIRR data by using our methodology.

The simulation environment for the reported experiments is based on JADE running on JVM 1.5. In JADE, each NEs and the OSS node run as software agents implemented as separate Java threads.

7.1 Centralized OK-PING

The OK-PING application in real mobile network has a centralized architecture with the OSS node initializing the communications towards to NEs and then collecting their replies. If a NE does not reply before a time out period, a fault management procedure is activated. The abstract code of the application parts running on the OSS node and on each NEs follows:

```
// Location: OSS node; Application: OK-PING.
    Every t seconds do {
        for any NE in the mobile network
            send a OK-PING request
        NEset={the set of all the NEs in the network}
        while ((some timeout is not exceed) and
                not(empty(NEset))) {
            collect OK-PING replies from any NE
            remove from NEset the NE who replied
        }
        if not(empty(NEset))
            {some NEs cannot be reached,
        start fault management process }
    }
```

```
// Location: NE; Application: OK-PING.
    Every t seconds {
        prepare status data
        wait to receive a OK-PING request from OSS
        send OK-PING reply to OSS with status data
    }
```

For our work, the message size of a OK-PING request is 10 bytes, an OK-PING reply consists of 60 bytes and the interval of time, t, is set to 60 seconds. The values of these parameters in real mobile networks depends on the chosen configuration.

We are interesting in measuring PIRR data for the concentrated OK-PING application when different numbers of NEs are in the network. Those data will provide information about the scalability of the application. We will also use the collected PIRR data to compare them against a different implementation of the same application, reported below, where a more distributed architecture is adopted.

By looking at the code, we can calculate that for each NE two messages have to be exchanged (the request and reply ones) between the OSS node and the NE to accomplish the task.

7.2 Decentralized OK-PING

This version of the OK-PING application is experimental and is not deployed on real mobile network. In this implementation, each single NE has the liability to periodically update the OSS node about its status by using a 'fire and forget' approach. If no communication is received by every NEs before a timeout expires, than a fault management procedure is activited for those NEs who have not sent the 'ok-ping' message. In this application, the OSS node periodically listen to the incoming communication and update its network status.

The abstract code of the application part running on the OSS node and on each NEs and follows:

```
// Location: OSS node; Application: OK-PING.
    Every t seconds do {
        NEset={the set of all the NEs in the network}
        while ((some timeout is not exceed) and
                not(empty(NEset))) {
            collect OK-PING replies from any NE
            remove from NEset the NE who replied
        }
        if not(empty(NEset))
            {some NEs cannot be reached,
            start fault management process }
    }
```

```
// Location: NE; Application: OK-PING.
    Every t seconds {
        prepare status data
        send OK-PING message to OSS with status
    }
```

For our study, the size of a OK-PING message is 60 bytes and the interval of time, t, is set to 60 seconds.

Also by examining the code, it appears that for each NE only one message is needed to inform the OSS node that the NE is working properly.

7.3 The PIRR methodology applied to the centralized OK-PING

Ten experiments per each network configuration (using a simulated network of 20, 40, 80 NEs) have been run to average the collected PIRR data by using the concentrated version of the OK-PING application.

Applying the PIRR methodology
In order to collect some of the PIRR data we are interested in, we have used NetBeans 5.5 Profiler. The profiler application allows for measuring the CPU time used by every NEs and by the OSS in networks with a different number of NE elements.

We then performed a linear interpolation in order to estimated how much the OSS node's CPU usage would vary depending on the number of NEs present in the network. This simple calculation, allows us to represent PIRR data in a parametric form which is useful when predictions for unseen condition have to be made.

In order to measure the memory utilization, the information provided by the profiler is not directly useful as only the total memory used (Heap and Stack) by the JVM is reported. So we have estimated the memory used by each NE node to run the OK-PING application, by averaging the total memory variations, when running the simulator with different NE load, over the number of NEs in the network. In practice, we used the following formula:

$$((TMU(80) - TMU(40))/40 + (TMU(40) + \\ - TMU(20))/20)/2$$

where TMU(n) stands for Total Memory Used (averaged over the 10 run experiments) when the number of NEs in the network is n.

The bandwith required by the application has been estimated by performing a code analysis, calculating the total byte size of the messages exchanged over a period of time and dividing the total byte size per the seconds in the time period. A communication set up overhead, whose value we were not able to measure, has to be added.

The storage requirement have been calculated by summing up the size of the compiled java file (class files) that make up the application plus estimating the size of temporary file used by the applications. No temp files are used by the studied applications.

Reporting PIRR data
In the table below, PIRR data for a configuration with 40 NEs reporting to the OSS are reported in a parametric format:

	OSS (40 NEs in the net)	1 NE
CPU	0.02% * 40	0.02%
Memory	α	35 KB
Bandwith	6.4 B/s + (40 * conn set up) +	0.16 B/s + conn set up +
	60 B/s + (40 * conn set up)	1 B/s + conn set up
Storage	9 KB	6 KB

Note that the Memory need (α) for the OSS node cannot be estimated. We know, by the code analysis, that the OSS node will use a fixed amount of memory for its internal objects and that amount will not change during runtime. If we run the system with only the OSS system to measure the total heap memory use (inclusive of JADE environment) we notice that it stays constant over time and does not exceed 700KB. That means that the memory usage for the OSS node is stable and its upper bound is 700KB. Further if we inspect the application code running on the OSS and we compare it with the code running on a NE, then a better estimate for an upper bound for the OSS node memory need is 50KB.

It is important to note that the PIRR data are reported in a parametric format obtained by combining the code analysis step with the experimental phase of the PIRR methodology. We will then be able to use the parametric data to estimate PIRR values for the application when it might run in different configuration settings.

7.4 The PIRR methodology applied to the distributed version of OK-PING

Again by using the distributed OK-PING application, we run ten experiments per each network configuration (using a simulated network of 20, 40, 80 NEs), and the averaged PIRR data have been collected.

In the table below, parametric PIRR data for a mobile network with 40 NEs reporting to the OSS are shown:

	OSS (40 NEs reporting to it)	1 NE
CPU	0.02% * 40	0.02%
Memory	β	35 KB
Bandwith	60 B/s + (40 * conn set up)	1 B/s + conn set up
Storage	5 KB	5 KB

As in the previous subsection, the reported values are parameter based, so they show the relationships between the application cost drivers and the number of NEs in the network. As obvious, they could be used to estimate PIRR data for the OK-PING, distributed version, under different conditions.

7.5 Two ways to exploit PIRR tables

PIRR tables, associated with information about the network configuration, could be used

1. to compare alternative versions of an application at the design time to decide which one to further engineer, or

2. to decide at runtime if or not to run a distributed application by dynamically estimating its resource requirements.

As an example of the first use of PIRR tables, comparing alternative implementation of the same application, the results reported in the above tables could be used to support the case to substitute the decentralized version of the OK-PING application for the concentrated one. The decentralized version requires less communication bandwith, as less messages are exchanged, and thus could help reducing the amount of network management traffic running through the mobile network. Also the storage requirements for the two versions, show that the distributed version can be stored in a smaller bytecode file.

Coming to the second use of PIRR tables, any application to be deployed into the mobile network could have associated a parametric table calculated a priori in a simulator. Then depending on overall load condition on the mobile network, a controller process could estimate if the mobile network has enough resources to run the application, and decide if to grant permission to execute to the application or postpone its execution when enough resources are available. This usage of the parametric PIRR table could support the move from a centralized and static NM approach, such as the one currently used in commercial systems, to a distributed and dynamic one where the NM system could deploy across the mobile network just the right combination of NM applications, from a portfolio of many, depending on the available resources and with a reasonable certainty that they would run as required and not adversely affect the network.

The control at runtime over which applications are allowed to run or not, could allow to reduce both the economic investment in network hardware and bandwith, and the interference in the core network between payload traffic and NM traffic.

8 Conclusions

In the paper we described the proposal and application of the PIRR methodology. The PIRR methodology can be used to measure and estimate computational resource requirements for distributed applications used in mobile networks. Moreover, we described and commented the experiments performed in a simulated network. We also show how to summarize application performances in parametric PIRR tables that can be used to estimate the resource requirement for the application in different network workloads. We also commented that PIRR data could be useful to evaluate at runtime when to deploy or delay execution of an application in mobile network to avoid overloading it.

References:

[1] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* Wiley, 2007.

[2] Cisco Systems. Using cisco network planning solution for capacity planning and optimization. Technical report, Cisco Systems, 2006. http://www.cisco.com/en/US/products/ps6363/products-white-paper0900aecd804b9201.shtml.

[3] F. Desbrandes, S. Bertolotti, and L. Dunand. Opnet 2.4: An environment for communication network modeling and simulation. In S. for Computer Simulation, editor, *European Simulation Symposium*, pages 64–74, 1993.

[4] JADE authors. *JADE - Java Agent DEvelopment Framework*, 2007. Available at http://jade.tilab.com/.

[5] S. R. Kunkel, R. J. Eickemeyer, M. H. Lipasti, T. J. Mullins, B. O'Krafka, H. Rosenberg, S. P. VanderWiel, P. L. Vitale, , and L. D. Whitley. A performance methodology for commercial servers. *IBM Journal of Research and Development*, 44(6):851–872, 2000.

[6] P. B. Luc Hogie and F. Guinand. An overview of manets simulation. *Electronic Notes in Theoretical Computer Science*, 150:81–101, 2006.

[7] ns2. The network simulator. ns-2., 1999. http://www.isi.edu/nsnam/ns.

[8] R. Vilalta, C. Apté, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive algorithms in the management of computer systems. *IBM Systems Journal*, 41(3):461–474, 2002.

[9] E. Weyuker and A. Avritzer. A metric for predicting the performance of an application under a growing workload. *IBM Systems Journal*, 41(1):45–54, 2002.

[10] X. Zeng, R. Bagrodia, , and M. Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.