

Understanding Scalability Issues for a Distributed Simulation Environment Using Intelligent Coordinated Entities

April R. Crockett, Rodger Maarfi, S. Ramaswamy, E. L. Brown, M. Rogers

Software Automation and Intelligence Laboratory, Department of Computer Science

Tennessee Technological University, Cookeville, TN 38505.

Email: srini@acm.org, Phone: (931)-372-3691

ABSTRACT:

Distributed applications are no longer an exception to software architecture design. Scalability is a strong motivation of many organizations to implement a distributed application. This paper presents a scalability case study using a distributed highway simulation environment. We report on improving an application's scalability using different measures related to the simulation architecture. Specifically, we study thread reorganization and network communication reorganization issues and their effect on scalability. We use performance and efficiency as the main tests for application scalability considerations. We use CPU overheads (measured in terms of CPU utilization), collision detection, thread sleep cycle, and network efficiency as the measures for testing performance and efficiency. We adopt a scheme with successive modifications, so that we can learn from each individual effort to improve scalability and judge collectively improvements.

1. INTRODUCTION

This paper is part of a project called Intelligent Coordinating Entities (ICE) that develops methods, tools, and methodologies that aid in the design of software systems using scalable, basic underlying communication-centric structures. We have previously presented a distributed environment for an automated highway system using a hierarchy of communication mechanisms to resolve advanced traffic situations [33]. It was developed to study practical issues related to ICE-based design and development as well as understand various communication paradigms. This paper addresses the following issues of scalability: (i) software applications' need to be concerned with scalability, (ii) testing the scalability of an existing software system, and (iii) small design improvements to an existing system that effectively address the issue of scalability.

Scalability implies the capacity of a software system to be able to handle increasing load or demand, i.e. the system should both retain, and preserve the quality of, its basic services. The main problem with software scalability is that there are no clear definitions or predefined ways to develop scalable software systems. Previous research pertaining to our work is illustrated in Figure 1. Most of our research has thus far focused on other software systems that had the scalability requirement and observing how it was accomplished. In our research to improve software scalability, we found that there is not one standard acceptable solution to scalability. However, we did find that

many existing software systems used similar methods to scale their applications.

This paper is organized as follows: Section 2 presents a survey of various software system domains that have addressed the issue of software scalability. Section 3 briefly presents our case study of a distributed highway simulation environment. Section 4 presents two different techniques adopted to improve the scalability of the application. The distributed highway simulation environment is already developed as a three-tiered architecture and hence the effect of a tiered architecture on scalability is not addressed in this paper. The benefits of a tiered architecture are reported in [2]. Section 5 concludes the paper.

2. Literature Survey

2.1 Scalability, Internet and Related Issues

A critical concern for those responsible for creating and maintaining large, high-volume web sites is scalability. Java Server Pages (JSP) and other Internet related software has been developed while focusing on scalability. For instance, the scalability of Java servlets impacts the scalability of JSP applications. The scalability of servlets is achieved because it uses a multithreaded approach to handling simultaneous requests [1]. Multithreading allows multiple requests to be easily handled by distributing the workload. Scalability is not only applicable to the Internet, but to every different kind of software. Other Internet related software includes a scalable naming and location service (NLS), e-business infrastructure and Entity Java Bean (EJB) Applications. A NLS must be scalable to make the web more scalable. NLS must be able to support a very large number of names, with associated objects that can potentially be located anywhere in the world. NLS is scalable because it is based on a dynamically configured, distributed search tree, with a fat-tree based topology at the global layer and spanning trees at the local layer [10]. An e-business depends on their company's ability to design an infrastructure that produces the measures of high performance, scalability, and reliability expected to support the business objectives for revenue and customer satisfaction. Scalability techniques normally involve the use of multi-tiered architectures. Scaling a multi-tiered infrastructure from end to end means managing the performance and capacities of each component within each tier. The basic objective of scaling a component of the system is to shift or reduce the load on the individual components and therefore increase efficiency [21]. Dynamic content in web sites is increasing rapidly, which creates a need for maintainable, reliable and scalable platforms to host these sites. The Java 2 Platform Enterprise Edition (J2EE) addresses some of these issues [30]. Many companies are now using Entity Java Bean (EJB) architecture to implement their

business applications since it provides a higher-level programming model to take advantage of the Java platform's features to improve efficiency [31].

2.2 Scalability and Distributed Systems

Parallelism in distributed systems may actually boost performance [19]. Scalability is also strong motivation to implement a distributed database system. A distributed database system that supports heterogeneous local DBMSs means supporting the concept of a gateway and when appropriately implemented, it can facilitate scalability. In [26], the authors discuss assessment of the performance characteristics of distributed software architectures using the Software Performance Engineering (SPE) approach. The SPE process begins early in the software life cycle and uses quantitative methods to identify a satisfactory architecture and to eliminate those that are likely to have poor performance. Scalability of Asynchronous Transfer Mode (ATM) network management system architecture, based on distributed artificial intelligence schemes and multi-agent systems, is discussed in [25]. The simulations are distributed applications using processes to simulate each ATM node. There have been some attempts to design a method to evaluate scalability in a distributed system, and even some specific ideas for the scalability evaluation of multi-agent systems, but the final evaluation of a concrete system design and the comparison with other designs is still very difficult [27]. Parallel and distributed discrete-event simulation (PDES) is a critical technology for an important class of very large complicated simulation models. In [28] scalability, locality, partitioning and synchronization in PDES is discussed and constraints on how to perfectly balance workload is provided. These constraints include maintaining workload balance, increasing model complexity, and simultaneous growth in model and architecture.

2.3 Scalability and Information Sharing and Retrieval

Another class of software systems concerned with scalability is information sharing and retrieval applications. The scalability of information retrieval systems has become increasingly important due to the rapid growth in the amount of collected information; however, there are few scalable systems available such as TELLTALE [3]. TELLTALE takes scalability into consideration with good performance by integrating new data structures and gamma compression. According to [12], there is an ever-increasing need for a distributed algorithm that would allow peer-to-peer applications to scale to a large community of users. The main difficulty in designing such algorithms is that currently, very little is known about the nature of the network topology on which these algorithms would be operating. The authors used Gnutella, a distributed information sharing technology based on a peer-to-peer model, as a case study to present strong small-world characteristics and a power-law distribution of node degrees to improve scalability.

2.4 Scalability and File Systems

File systems scalability is another area of work. File systems for workstations and servers must be able to scale capacity and performance due to the size of the file systems. File system scalability is defined as the ability to support very large file systems, files, directories, and numbers of files while providing comparable I/O performance to smaller file systems [13]. Mechanisms for managing large files, large numbers of files, large directories, and very high performance input/output must exist [9]. One file system that claims to be scalable is the XFS file system that uses mechanisms for scalability such as on-disk data structures and B+ trees in place of the linear file system structures.

2.5 Summary

In every different type of application in question, scalability is solved in a number of different ways. However, reorganizing memory, reducing memory usage, redistributing load, increasing communication efficiency, improving thread management and increasing parallelism can improve scalability for most distributed applications. In the following sections, we look at different ways to improve scalability for our particular case study.

3. CASE STUDY

This case study was originally designed as a distributed environment for an automated highway system using a hierarchy

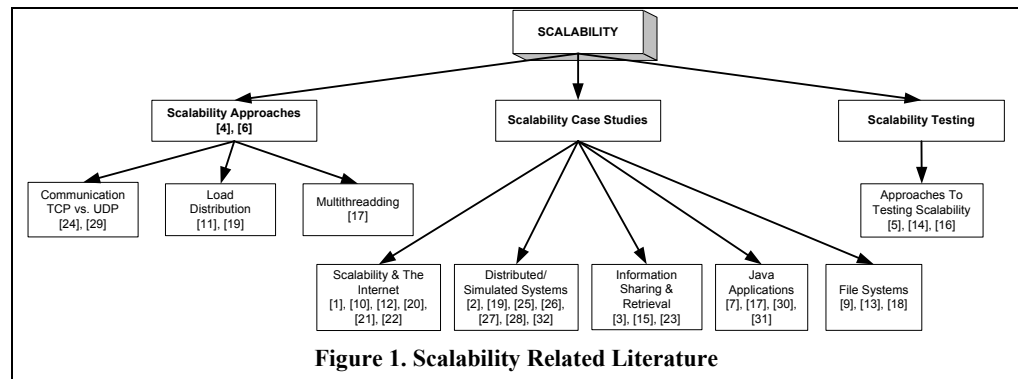


Figure 1. Scalability Related Literature

of communication mechanisms to resolve advanced traffic situations [33]. It was mainly developed to study practical issues related to Intelligent Coordinating Entities (ICE)-based design and development as well as understand various communication paradigms.

The simulation is currently implemented as a client/server, peer-to-peer system in JAVA. Vehicles communicate through a linked graph called a 6-way dynamic socket mesh (DSM) and through a 3-layer intelligent control and coordination structure consisting of a lower level vehicle control layer, a segment control layer and a central simulation controller. The three level tiered architecture provided for (i) Localization of control information that reduced system overheads, (ii) Localization of immediate environment information that reduced network traffic, (iii) Pseudo-globalization of normal operations policy whereby some information such as platoon management can be better applied at the segment level, and (iv) globalization of emergency policy for intelligent road navigation in case of emergency situations, such rerouting of traffic for evacuations.

In our effort to improve the scalability of the simulation environment, we have addressed two basic implementation issues that have a major impact on scalability. The first issue was to

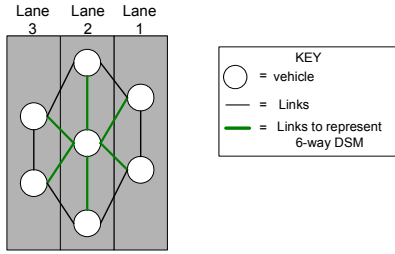


Figure 2. Six way Dynamic Socket Mesh (DSM)

design a more efficient multi-threading mechanism.

Originally, each vehicle implementation ran several threads. Minimizing the number of threads reduced the time required for the application to switch

between threads. The effective organization of threads also prevented CPU bound logic from waiting on I/O operations such as network sending and receiving. Another issue was to make communication more efficient. As mentioned earlier, a six-way DSM was used in the original system and TCP/IP was the communication protocol. To improve scalability we have re-implemented the system using UDP instead of TCP. By using UDP broadcasts, all surrounding cars could be simultaneously notified of an event. Since cars kept track of neighboring cars, tracking message acknowledgements was simplified.

We report on improving the application's scalability in this paper using two different measures related to the simulation architecture. We use performance and efficiency as the main tests for system scalability considerations. We use CPU utilization, thread sleep cycle, and network efficiency as the measures for testing performance and efficiency. We adopt a scheme with successive modifications, so that we can learn from each individual effort to improve scalability as well as judge collectively improvements at the end. In the following sections, we will present the issues in more detail that have had a significant impact on scalability.

3.1 Communication Improvements

The original system used the TCP/IP protocol to communicate basic car information in a six-way Dynamic Socket Mesh (DSM), which represents the nearest car at that location. The worst-case scenario for any point in time is a vehicle having six surrounding links, shown in figure 2. This means each time a decision needs to

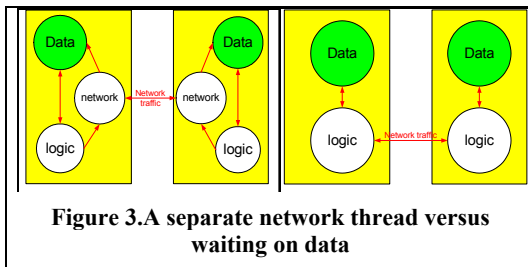


Figure 3. A separate network thread versus waiting on data

be made by a particular vehicle or vehicles, as many as six cars might need to be consulted when attempting communication. Keeping track of the data and responding to events instead of polling causes network efficiency to be dramatically increased. By using UDP broadcasts, six cars can be notified of an event occurring at once, and because cars keep track of what cars are connected to it, the broadcasting car knows how many acknowledgements need to be received.

3.2 Multithreading Issues and Data Organization

On the original system, each car ran several threads. To run many vehicles on each computer, threads should be minimized. However, threads that contain I/O bound operations, such as network sending and receiving, should be separated from threads that are CPU bound. Figure 3 shows how separating I/O threads from the rest of the vehicle's logic improves performance. We have modified the original system with organized, minimized threads. The overall reduction in the number of threads increases efficiency because it minimizes the thread scheduling overhead.

One of the weaknesses of Java is lack of event driven IO. Using event driven IO mechanism may have further reduced the number of threads involved and hence the thread switching overheads.

4. TESTING SCALABILITY

To accurately test scalability, several different factors must be measured, depending on the system architecture. Performance and efficiency are the main tests to measure if the architecture of a system is scalable. First, to measure the different changes to the simulation that were made, we applied the modifications to the current system one at a time so that each impact could be accurately measured. The first modification to the simulation was changing from unorganized threads to a reduced number of organized threads. The second modification was using UDP instead of TCP/IP. At each change in the simulation and before any of the changes, we test the scalability by the measures described below.

We use four different ways to measure the effectiveness of the changes made including reducing CPU overheads (measured in terms of CPU utilization), number of collisions between vehicles, time that the threads are asleep, and network efficiency. The remainder of this section describes each of these measures in detail. The following sections present measures of testing that are relevant to the highway simulation. We use collisions between vehicles as the measure to determine the upper bound.

4.1 Testing Java CPU Utilization

The Java CPU utilization test (measuring reduction in CPU overheads) is a measure of the percentage of the total processor time a PC spends retrieving data from the disk subsystem. The CPU Utilization we test is dependent on Java. The goal in able to make software scalable is to minimize CPU utilization. It is best if the simulation does not require much CPU usage. If the simulation requires 100% of the CPU, then it will not scale, and problems such as collisions or even a crash in the simulation will result. We tested all three systems for CPU utilization. The results for the original system are shown in the graph in Figure 4a, whereas the results for the organized threads implementation are shown in Figure 4b. The number of vehicles in Figure 4a was about 120 cars and the number of vehicles in the second graph with the organized threads jumped to 224 cars (i.e. 186%). This proves that better thread organization can make software more efficient and therefore contributes positively to software scalability.

4.2 Testing Threads

To test the threads themselves, we measure thread latency. If they are asleep for a very small amount of time, then it does not affect scalability, but if they are asleep for too long, problems could result (effect CPU utilization) and the simulation will not scale

effectively. The graphs in Figure 4c and Figure 4d represent our tread latency test. On the original system, threads stayed asleep fairly consistently for the same amount of time until the number of cars increased to 108. After that, the threads started staying asleep for an entire second or more. After we organized the threads in the system, the highest number that the threads ever stayed asleep was only .3 seconds (3/10ths of the original system)! This also indicates that thread latency affected scalability even more than unnecessary CPU overheads(measured in terms of CPU utilization).

4.3 Testing Network Efficiency

The main scalability factor was communication efficiency. Figure 4c and Figure 4d illustrate this scenario. Testing network efficiency involves testing the time that it takes to send data and get it back (average ping). If the data in the simulation is not received or sent by the correct parties, then the vehicles will have incorrect information. This will result in collisions between vehicles. The network efficiency test is most important when we change the communication over the network from TCP/IP to UDP. Our results from testing the original and organized thread systems proved to us that our main scalability problem lies in our communication protocol. Figure 4e shows the poor network efficiency of the original system and Figure 4f shows the much improved network efficiency of the UPD system. The latency of the UDP system had the highest latency of less than half (450 milliseconds) of the highest latency of the original system (1000 milliseconds). On the UDP system, we were able to run over 350 vehicles before experiencing collision problems, i.e, improvement of about 296% from the original system and 158% over the reorganized threads implementation.

4.4 Testing Results

Through a simple case study we have shown that network efficiency, optimization of threads, and CPU overheads are important with respect to scalability. Table 1 summarizes the results. As observed, we were able to double the number of vehicles when implementing both the organized thread system and the UDP system combined (210% on a single machine and 141% on four machines).

5. CONCLUSIONS

In this paper we have identified and reported on two basic implementational ideas to improve scalability and three different metrics for measuring software scalability. These include CPU overheads, thread latency and network efficiency. We have also shown that simple design issues such as better thread organization in a multithreaded programming environment and using UDP instead of TCP/IP as a communication protocol may greatly affect scalability issues. While the reorganization of threads may be an easier design issue on many applications that can be sufficiently reasoned by a programmer, the use of UDP over TCP/IP should be subject to more scrutiny based on the reliability needs of the application.

6. REFERENCES

[1] Duane K. Fields and Mark A. Kolb. "Scalability in JSP," available at http://www.webreview.com/2001/02_16/developers/index04.shtml as of September 12, 2002.

[2] Bernard P. Zeigler. "Scalability Considerations in Measuring Intelligence: Insights from Modeling and

Table 1. Number of Vehicles in each testing simulation

	Original System	Organized Threads	UDP System
Testing on one machine	120	224	252
Testing on 4 machines	252	288	356

Simulation," 2002 Performance Metrics for Intelligent Systems, August 2002, National Institute of Standards and Technology, Gaithersburg, MD, August 13-15, 2002.

[3] Ethan Miller, Dan Shen, Junli Liu, and Charles Nicholas. "Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System," available at <http://jodi.ecs.soton.ac.uk/Articles/v01/i05/Miller/miller2.pdf> as of September 12, 2002.

[4] Kyle Gabhart. "Disciplined Software Engineering," available at <http://216.239.39.100/search?q=cache:wyrpST3h8kcC:www.csm.ornl.gov/~v8q/Homepage/Class/PSP/Lectures/Final%2520Summary.PPT+carnegie+mellon+university+disciplined+software+engineering+-+lecture+11&hl=en&ie=UTF-8> as of September 19, 2002.

[5] Kyle Gabhart. "Flexibility and Scalability," available at http://gethelp.devx.com/techtips/java_pro/10MinuteSolutions/Gabhart10min02-4.asp as of September 19, 2002.

[6] Matt Liiotta. "Scalability's New Meaning," available at http://www.bayprofessional.com/articles/view_article.php?id=1&pid=1 as of September 19, 2002.

[7] Nengendra Nagarajayya and J. Steven Mayer. "Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory," available at <http://wireless.java.sun.com/midp/articles/garbage/> as of September 19, 2002.

[8] Diane Olson and Robert Ray. "Version 9: Scaling the Future," available at <http://www.sas.com/rnd/base/topics/scalingfuture/scalingfuture.pdf> as of September 23, 2002.

[9] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. "Scalability in the XFS File System," available at http://oss.sgi.com/projects/xfs/papers/xfs_usenix/ as of September 24, 2002.

[10] Y. Charlie Hu, Daniel A. Rodney and Peter Druschel. "Design and Scalability of NLS, a Scalable Naming and Location Service," available at <http://www.cs.rice.edu/~druschel/publications/nls.pdf> as of September 25, 2002.

[11] Ian Foster. "Designing and Building parallel Programs," available at <http://www-unix.mcs.anl.gov/dbpp/> as of September 26, 2002.

[12] Mihajlo A. Jovanovic, Fred S. Annexstein, Kenneth A. Berman. "Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella," available at <http://www.ececs.uc.edu/~mjovanov/Research/paper.html> as of September 27, 2002.

- [13] Philip Trautman and Jim Mostek. "Scalability and Performance in Modern File Systems," available at http://oss.sgi.com/projects/xfp/papers/xfp_white/xfp_white_paper.html as of September 30, 2002.
- [14] "Scalability" available at http://search390.techtarget.com/sDefinition/0,,sid10_gci212940.00.html as of October 1, 2002.
- [15] Sean Gallagher and Steve Gillmor. "The Scalability Factor," available at <http://www.informationweek.com/730/com.htm> as of October 2, 2002.
- [16] Gordon Lyon. "Comparing Two Distinct Approaches to Scalability Testing," *Calculateurs Paralleles. Volume 8 - n 3/1996, pages 329-335*.
- [17] Jean-Luc. "Scalability – Load-Balancing – Fault tolerance," available at <http://java.apache.org/jserv/howto.load-balancing.html> as of October 7, 2002.
- [18] Nikos Drakos. "Scalability of Branch-and-Bound and Adaptive Integration," available at <http://www.cs.wmich.edu/~parint/trr/papers/pdpta01/html/pdpta01.html> as of October 8, 2002.
- [19] Rick Noel. "Scale Up in Distributed Databases – A Key Design Goal for Distributed Systems," available at http://www.cs.rpi.edu/~noel/distr_scaleup/distributed.html as of October 9, 2002.
- [20] Jon Siegel. "The Right Way To Scale," available at <http://www.internetweek.com/sotry/INW20010723S0001> as of October 10, 2002.
- [21] Willy Chiu. "Design for Scalability – An Update," available at <http://www7b.software.ibm.com/wsdd/library/techarticles/hvws/scalability> as of October 21, 2002.
- [22] Intelligent Enterprise Magazine. "New Apps Make Scalability Paramount," available at http://www.intelligententerprise.com/db_area/archives/1999/991602/editpage.shtml/scalability as of October 22, 2002.
- [23] Michael Otey. "The Route to Scalability," available at <http://www.sqlmag.com/Articles/Index.cfm?ArticleID=16121> as of November 4, 2002.
- [24] IBM. "TCP or UDP?," available at http://www.transarc.ibm.com/Support/dce/general/tcp_and_udp.html as of November 5, 2002.
- [25] Pere Villa and Josep L. Marzo. "Scalability Study and Distributed Simulations of an ATM Network Management System Based On Intelligent Agents," available at http://eia.udg.es/~perev/pdf_versions/spects_v07.pdf as of November 8, 2002.
- [26] Connie U. Smith and Lloyd G. Williams. "Performance and Scalability of Distributed Software Architectures: An SPE Approach," available at <http://www.perfeng.com/papers/pdcp.pdf> as of November 12, 2002.
- [27] Sandeep K. Singhal and David R. Cheriton. "Using Progression Aggregations to Support Scalability in Distributed Simulation," available at http://citeseer.nj.nec.com/cache/papers/cs/1909/ftp:zSzzSzftp_dsg.stanford.edu/SzpubzSzpaperszSzprojections.pdf as of November 13, 2002.
- [28] David M. Nicol. "Scalability, Locality, partitioning and Synchronization in PDES," available at <http://delivery.acm.org/10.1145/280000/278010/p5-nicol.pdf?key1=278010&key2=6728819301&coll=portal&dl=ACM&CFID=6141489&CFTOKEN=65875203> as of November 15, 2002.
- [29] Paul N. Leroux. "Redefining Software Scalability for the Network Infrastructure," available at http://intel.com/pca/developernetwork/solutionsjournal/spring_02/pdf/qnx_redefining.pdf as of November 21, 2002.
- [30] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. "Performance and Scalability of EJB," available at http://www.cs.rice.edu/CS/Systems/DynaServer/perf_scalability_ejb.pdf as of November 22, 2002.
- [31] Adam Rybicki. "Achieving Performance and Scalability with Enterprise JavaBeans," available at http://e-serv.ebizq.net/shared/goldclub.jsp?rybicki_1b.html as of November 23, 2002.
- [32] Brian Blum, Tian He, Yvan Pointurier. "MAC Layer Abstraction For Simulation Scalability Improvements," available at <http://www.cs.virginia.edu/~bmb5v/cs851/report.pdf> as of December 1, 2002.
- [33] R. Maarfi, E. L. Brown, S. Ramaswamy, "A Three-Tier Communication and Control Structure for the Distributed Simulation of an Automated Highway System", 2002 *Performance Metrics for Intelligent Systems*, August 2002, National Institute of Standards and Technology, Gaithersburg, MD, August 13-15, 2002

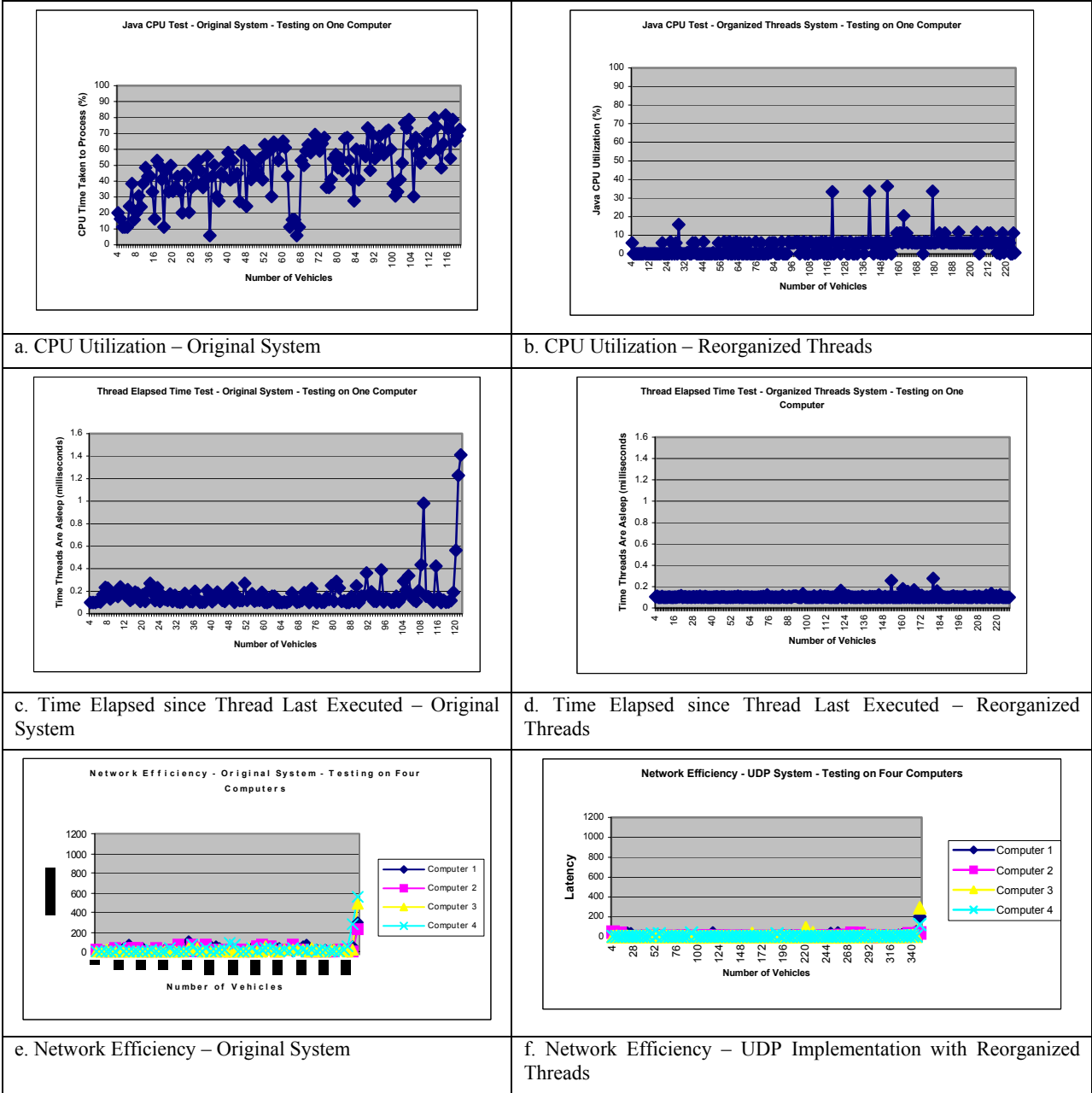


Figure 4. Observed Results