Federated Clusters Using the Transparent Remote Execution (TREx) Environment

Richert Wang[1]
University of California, Irvine
442 Computer Science Building
Irvine, CA 92697-3435
rkwang@ics.uci.edu

Enrique Cauich
University of California, Irvine
442 Computer Science Building
Irvine, CA 92697-3435
ecauichz@ics.uci.edu

Isaac D. Scherson
University of California, Irvine
442 Computer Science Building
Irvine, CA 92697-3435
isaac@ics.uci.edu

***Abstract** - Due to the increasing complexity of scientific models, large-scale simulation tools often require a critical amount of computational power to produce results in a reasonable amount of time. For example, multi-system wireless network simulations involve complex algorithms of traffic balancing and communication control on large geographical areas. Moreover many of these intensive applications are designed for single sequential machines and large sums of money are spent on purchasing powerful servers that can give results in a satisfactory amount of time.*

The aim of this paper is to introduce a general-purpose tool, dubbed Transparent Remote Execution (TREx), which avoids resorting to expensive servers by providing a cost effective, high performance, distributed solution. TREx is a daemon that dynamically exploits idle operational in-use workstations. Based on elaborate rules of computational resource management, this daemon permits a master to scan workstations within a pre-defined subnetwork and share the workload among the least occupied processing elements. It also provides a clear framework for parallelization that applications can exploit. By providing a simple way of federating computational resources, such a framework could drastically reduce hardware investments.

1. Introduction

Imagine a typical institutional network with many workstations interconnected via a TCP/IP network. The majority of these workstations are not utilizing the full potential of their processors. For example, a secretary required only to use an office program and internet browser does not consume the full amount of the workstation's processor or memory. These minimally used computers are considered *clerical stations* (CS). In this institutional network, only a few computers are used for scientific computing (large-scale simulations and complex scientific modeling) requiring more resources than what a personal computer provides. These workstations that need more resources for intensive computational programs are considered *engineering workstations* (EW). Normally, the number of clerical stations is greater than the number of engineering workstations.

The Transparent Remote Execution (TREx) tool was designed to exploit these clerical stations so engineering workstations can use idle CPU cycles to perform useful computations. The basic components within TREx are the master daemon, which run in workstations with heavy workloads, and the slave daemon, which run in under-loaded workstations. Master workstations can delegate workload to all registered slave workstations. Masters will send the executable (service), and any necessary input files, to its slaves. Slaves will remotely execute any service that the master sends it and send the output back to the master as if the program was run locally. All execution is done transparently and the user of the workstation is unaware of the remote processing occurring. A master may also be a slave and can either process its own workload that it delegates to itself or process workload received from other masters.

The content of this paper is the following: Section 2 will analyze the differences between TREx and other solutions introduced in the literature, making emphasis on the fact that this new environment is more transparent, easier to use, and takes the variability of computational loads in federated workstations into account. Section 3 presents the TREx architecture, which allows a dynamic federation of workstations defined by a distributed configuration file, which will be explained in more detail in Section 4. TREx also provides a clear framework for parallelization, which will be discussed in Section 4.3. Based on this framework, applications can be designed to exploit parallelization in a federated cluster. Section 5 summarizes experimental results using TREx and Section 6 concludes the paper.

2. Background

There has been a plethora of work regarding remote execution in distributed systems and all possess the same goal: distributing workload to workstations and exploiting parallelization through a high-latency, low-bandwidth interconnection network. A comparison of well-known distributed systems is shown in Table 1. [6] discusses that the

[1] TREx experiments and results were obtained during an internship at France Telecom R&D, 38-40 rue du General Leclerc Issy les Moulineaux, France

SYSTEM	OBJECTIVE	LIMITATIONS	EXISTING TECHNOLOGY	DISTRIBUTION TECHNIQUE
CONDOR (1988)	Scheduling system that distributes background jobs to idle machines.	RAM or HD space not incorporated in selecting idle machines.	T1 connection enabled (1.544 Mbps).	Procedure Calls
CORBA (1991)	Standards allowing the interoperability and portability of distributed Object Oriented applications.	Centralized Broker of services that can be provided. Non-distributed resource management	T3 Connection enabled (44.736 Mbps)	Object Lookup Service
PARALEX (1996)	Environment to hide the complexities of distributed computing to the user.	Centralized Resource Manger. Static description of resources. Does not distribute to "idle" workstations.	VBNS created. Dial-up (56 kbps) and partial T1 connection become widely available to the public	Sub-programs and Functions
JINI (1999)	Federations of networked components that can share their resources.	Services defined by servers, not clients. Does not distribute to "idle" machines.	ISDN technology reaches its peak.	Services / Object Lookup Service
GRID Superscalar (2003)	Reduce the complexity of GRID applications.	Centralized Broker of services. Services defined by servers. Does not distribute to "idle" machines.	Wireless Technology incorporated to LANs 802.11g (54 Mbps).	Services / Object Lookup Service

Table 1: System Comparisons

main concerns of distributed computing are latency, memory access, partial failure, and concurrency. It also emphasizes that developing technologies provide tools for distributed computing that were either unavailable or unfeasible in the past. With broadband connections to the internet becoming more advanced and available, the communication latency for TREx to send large execution input files to slaves and return large output files to masters is acceptable for practical cases. Services provided in TREx are independent pieces of code (functions, processes, programs, etc.) that do not share an address space among the other services. Partial failures are unavoidable in distributed systems, but TREx has mechanisms to recover from common faults such as failed network connection and workstations becoming offline. It also has safeguard mechanisms to prevent any remote application to exhaust a workstation's local resources and detect if remote execution failed. TREx provides a parallelization framework to handle concurrency of remote execution calls. This is covered in more detail in section 4.3.

Some approaches extend programming languages to allow parallelization. Examples of such systems are the MW framework, Amber, and ParaWeb [7, 4, 2]. However, all these approaches share the following drawbacks: they do not provide a simple tool allowing transparent utilization of workstations, they are programming language specific, and they do not provide an easy framework for parallelization. Condor [8, 3] is a scheduling system that identifies idle workstations to run background processes and is able to migrate processes to other workstations. However, Condor is only a scheduling system and does not provide a clear framework for parallelization, and Condor has a "central manager" that handles task distribution and resource management. In TREx, concurrency of workload execution can be organized based on the parallel framework and its resource management is fully distributed among the clients (masters) in the system. The responsibility to exploit concurrency within TREx is dependent on the application designer.

Three other well known distributed systems are CORBA [10, 11], JINI [10, 12] and GRID SuperScalar[1]. In these systems, the clients register their services to a *look-up server* (broker). The look-up server will forward a client's service request to the appropriate remote workstation. The clients in both systems are dependent on the look-up servers and the services offered by them. TREx gives the client full control of which services a remote workstation will compute by sending them the appropriate service files. Instead of the typical client / server model where the client requests a service to the server, TREx's masters tell the slaves the services it will execute.

Like TREx, JINI has a distributed resource manager (its components can register services to multiple look-up servers). However, CORBA's clients register to a single look-up server (broker), and the client will become unavailable if this broker

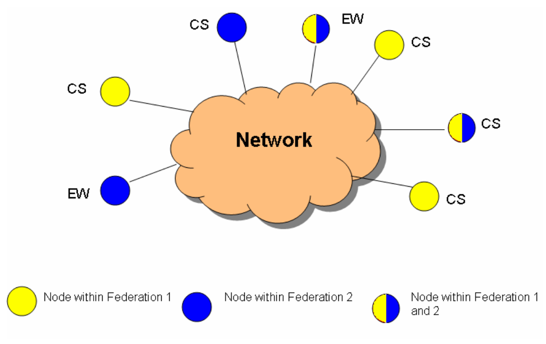
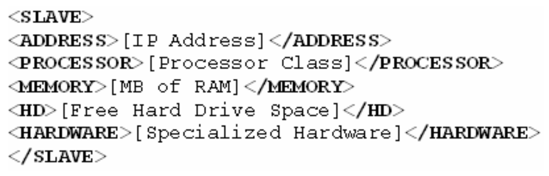


Figure 1: Example of Node Federations between CS and ES



```
<SLAVE>
<ADDRESS>[IP Address]</ADDRESS>
<PROCESSOR>[Processor Class]</PROCESSOR>
<MEMORY>[MB of RAM]</MEMORY>
<HD>[Free Hard Drive Space]</HD>
<HARDWARE>[Specialized Hardware]</HARDWARE>
</SLAVE>
```

Figure 2: IP.cfg Format

becomes unavailable. With a fully distributed resource manager, TREx slave workstation can still be accessed by other masters it is registered to.

The system most comparable to TREx is Paralex [5]. Both systems, as well as GRID Superscalar, use a library of services that can be organized in a Directed Acyclic Graph (DAG) to exploit concurrency. However, there are key differences between TREx and Paralex. First of all, TREx is designed to allow multiple masters to launch parallel applications simultaneously, where each master views a distinct partial subnetwork of the entire system. These subnetworks can overlap, i.e. a workstation can belong to many masters. Hence, the masters do not have access to the entire system like in Paralex, where information about the entire system is based on a centralized repository containing workstation information. Figure 1 shows an example of a federated network with nodes specifically belonging to one or more federations within the network. The advantage of assigning specific subnetworks to distinct masters is discussed in section 4.1.

Another key difference between Paralex and TREx is how slaves are selected during runtime. Paralex selects its slaves before execution starts for all services needed. TREx selects its slaves when a service is ready to be executed. The slaves of TREx are constantly being updated via registration messages with their available RAM, available hard drive space, processor type, and processor utilization percentage. The master records and modifies these values when deciding to delegate workload to a slave. Based on this information, a TREx slave can be unregistered from the system if its processor is busy or memory is not sufficient for remote execution based on a pre-defined threshold. A TREx master can choose any registered slave at any given time since only registered slaves are considered idle. The details of slave registration are given in section 3.3.

3. Architecture

This section will cover the architecture of TREx. Specifically, the master daemon, slave daemon, and the communication protocol between the two.

3.1. Master

The master daemon's primary function is to delegate workload to slaves and receive output from slaves. Some services may be dependent on the output produced by other services, therefore it is the job of the application developer to design the distributed application to have the correct flow of logic. In order for TREx to work, the master must have a script.mst file. The script.mst file contains service information, including minimum resource requirements needed, for all services that will be executed. The master parses this script file, obtains all necessary information about the services that will be launched, and invokes all available slaves within its cluster for remote execution. All services (executables, input files, libraries, etc.) must be present within the master.

Since any CS can be a slave, it is not guaranteed that a slave will have the necessary service files. Therefore, the master must check with the slave if it contains the appropriate service files and their integrity before sending the execution command. If an inconsistency is detected, the slave will receive an updated copy of the files that differ from the master's copy, even if the slave already possesses the file. The interface shows a list of all available slaves, all available services within a slave, and all files associated with the service.

The master must also use the IP.cfg file (discussed in Section 4.1). This file contains information about all slaves that are able to be registered to the master. Periodically, the master sends registration messages to all IP addresses in IP.cfg. If an IP address does not reply in a certain number of registration attempts, the communication is presumed dead and all services being executed at this location are rescheduled. All slaves will not be within a single IP.cfg file in order to prevent greedy masters from using the resources of all slaves in the system. Intelligent algorithms must be implemented on the master in order to decide which idle slave is most suitable to receive a certain service.

When an error in remote execution is reported to the master, the master must reschedule the service with another workstation. Rescheduling happens when a slave crashes, does not have anymore available resources, returns a negative exit code indicating a problem with execution, or communication between master and slave dies. Extended fault tolerance algorithms and implementation are topics for future work.

3.2. Slave

The slave daemon's primary functionality is to accept workload from the masters it is registered to, accept any service files if necessary, execute the workload given to it, and send the output back to the master. The slave must also constantly check if its resources do not fall under a certain threshold. If it does, then the slave will terminate remote execution, delete the directory where remote execution was taking place, and report an error back to the master. This is done to prevent a malicious or naïve application from crashing the system. The slave keeps track of all masters it is registered to and only accepts workload from its masters in order to prevent hacker attacks.

3.3. Communication

The communication between a master and slave has several steps. The first step is the slave registering to a master. The master will send registration request messages to all IP addresses found in its IP.cfg file in constant intervals. When a slave receives a registration request message, it will send a message back to the master with its available RAM, available hard drive space, and processor utilization. The master will decide if the slave is "idle" based on the slave's resource information. Only idle slaves will be registered to the master and all other slaves will be ignored until the next registration reply message is received with updated resource values.

Once the master has services to delegate, it checks to see if the slaves have the necessary files for execution. If they do not, then the master will send the missing (or outdated) service files. After the master sends all appropriate files to the slave, it will send the execution command. When the slave receives the execution command, it starts the execution. After execution is complete, the slave sends the output files to the master in a pre-defineda directory (sent to the slave with the execution command). The slave then sends the *exit code*, i.e. a numerical value indicating if execution was successful or not. When execution is successful, the master can then continue delegating any service ready for execution to its slaves. However, if an error was detetected, then the master must reschedule the failed service to another slave.

4. Resource Management

This section will discuss the resource management in TREx in more detail. The main parts of resource management in TREx are found in the IP.cfg and the script.mst file. Based on the data loaded from these files, intelligent algorithms can be developed to maximize performance.

4.1. IP.cfg

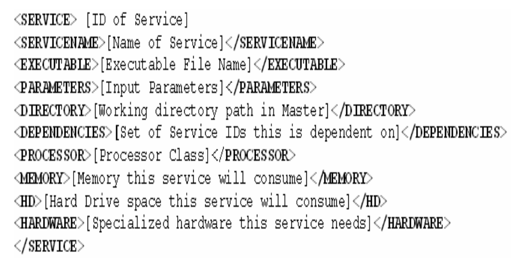
The IP.cfg file contains information about all slaves that can be registered into the federation. The format for each slave in the IP.cfg file is shown in figure 2. The information given includes the IP address of the slave, the processor type, free RAM, free hard drive space, and specialized hardware. The master now has an accurate description of what the slave is capable of. For example, certain services may require a graphics card or a certain amount of hard drive space. The master can choose the appropriate slave that satisfies the minimum requirements for a specific service.

An advantage to this IP.cfg file is the fact that a subset of slaves can be statically assigned to a master. Slaves must be federated in such a way so a single master cannot be "greedy" and use all of its slaves' resources while other masters have to wait. For example, imagine a cluster with five slaves and two masters where all five slaves are registered to both masters. If Master A has an application to run and requires all of its slaves' resources, then it can delegate its workload to all the slaves. When Master B wants to distribute its workload to the slaves, the slaves do not have any idle CPU resources, and therefore cannot register themselves to Master B. Master B must wait until the slaves finish processing their current workload before it can distribute its workload. However, Master A's application may take hours to compute, depending on the application. In this scenario, a deadlock occurs and the system becomes inefficient for Master B.

The way to prevent this is incorporating the responsibility of a *system administrator* to assign specific workstations in the master's IP.cfg file. Imagine the same cluster described above, however this time Slave A, B, and C are registered to Master A and Slave C, D, and E are registered to Master B. Slave C may not have available resources for Master B, but slave D and E are available since Master A cannot use them. This prevents Master A from being greedy and creating a deadlock.

4.2. script.mst

The script.mst file contains information about all the services that need to be executed. The format for each service in the script.mst file is shown in figure 3. Each service will be required to have a service ID, service name, executable file name, input parameters, working directory path, and a set of service IDs that it is dependent on. Other information



```
<SERVICE> [ID of Service]
<SERVICENAME>[Name of Service]</SERVICENAME>
<EXECUTABLE>[Executable File Name]</EXECUTABLE>
<PARAMETERS>[Input Parameters]</PARAMETERS>
<DIRECTORY>[Working directory path in Master]</DIRECTORY>
<DEPENDENCIES>[Set of Service IDs this is dependent on]</DEPENDENCIES>
<PROCESSOR>[Processor Class]</PROCESSOR>
<MEMORY>[Memory this service will consume]</MEMORY>
<HD>[Hard Drive space this service will consume]</HD>
<HARDWARE>[Specialized hardware this service needs]</HARDWARE>
</SERVICE>
```

Figure 3: script.mst Format

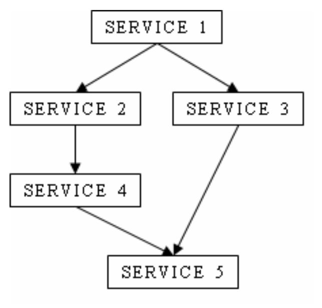


Figure 4: DAG Representing Execution Flow

representing the minimum requirements for the service, such as processor type, memory, hard drive and specialized hardware are not required. If these fields are not defined, then these values are set to pre-defined default values.

The reason why it is beneficial to include resource information such as a required processor, consumed memory and specialized hardware is for efficiency reasons and for protection against malicious or naïve script writers. A remote application may create an explosion of data that will overflow a slave's resources and crash it. TREx prevents this from happening. Using the information in the IP.cfg file, the script writer can specify the amount of hard disk space a service will take. If the application uses more space than what the script writer anticipated, the TREx slave will kill the job, delete all files produced from the job, and send an error message to the master. If the script writer does not know the exact amount of hard disk space required for an application, the TREx slaves will have a default free disk space and RAM size. Whenever the resources falls below this threshold, the slave will kill the application, delete all files associated with the current job, and send an error message to the master who will have to reschedule the job.

Before execution, the master loads the script.mst file and stores all services needed to be executed. The master will then compare its registered slaves' resources to the services that are ready to be launched, choose the best slave for processing, and send the services one at a time. The reason why services are sent one at a time rather than all at once is due to the fact that the slaves are occupied workstations. There is no way to predict when one workstation will be idle at any given time. Therefore, if a slave receives many services to process and becomes overloaded during runtime, then the overloaded workstation becomes a bottleneck for all other services waiting for its completion.

4.3. Parallelization Framework

The script.mst service definitions provide a clear framework for parallelization. The dependencies each service may have creates a flow of execution that can be represented as a DAG. Services that have no dependencies can be done in parallel if the resources are available. An example of how a DAG is constructed based on the definition of dependencies is shown in figure 4. As can be seen, the services defined in script.mst cannot be executed until their dependencies are finished. Service 2 and Service 3 cannot execute until Service 1 has finished. Service 4 can execute once Service 2 is done and Service 5 must wait for Service 3 and Service 4.

The use of this framework is dependent on the application developer. Since TREx is assumed to be used in an institutional network, which is normally a broadband connection, speedup gain must be significantly more than communication latency. The goal for high performance parallelization is to minimize the communication to computation time ratio. Therefore application developers must decompose these services to be *large-grain* so computation time is much longer than communication latency.

5. Experimental Results

TREx has been applied to France Telecom's Odyssee simulator [9] in order to run parametric experiments in a significantly shorter amount of time. Odyssee is a simulator that models large-scale heterogeneous networks. Interference, interactions between different systems, radio resource management, and QoS behavior are simulated and analyzed. Odyssee and its parametric tests were designed to run on a single workstation. The parametric tests were conducted by initially defining an amount of mobiles for a specific mobile service (i.e. sms, web browsing, ftp, etc.) in a given area; in this case the number of mobile services was three. Initially, the user defines the number of simulations to run and a number representing the rate of mobiles entering the network in each simulation. For example, imagine a test where the initial value is defined as 5 mobiles for mobile service A, the rate of increase is 5 mobiles entering the network per simulation, and three simulations are being requested. The first

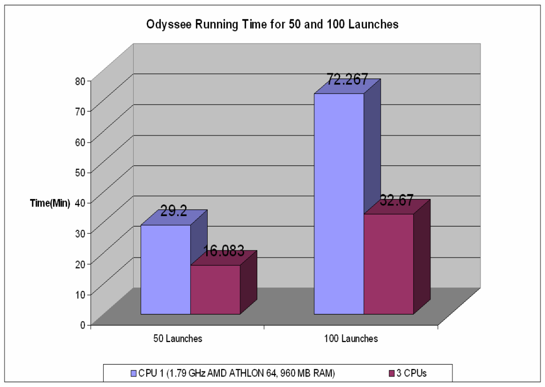


Figure 5: Results for 50 and 100 Simulations using Operational in-use Workstations

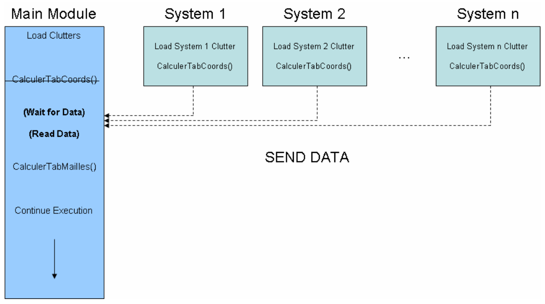


Figure 6: Modularization of the Persee Component

simulation would contain five mobiles, the second would have ten mobiles, and the third would have fifteen mobiles for mobile service A. Each simulation is considered a service within TREx. Many mobile services can be modeled and the effects of congested base stations can be analyzed within a specific area.

This preliminary experiment was broken up into two parts. First was the creation of the script.mst file, and second was parallelizing the workload. A PERL script was used to automatically generate the input files for execution, the working directories for each simulation run, and the script.mst file in the format shown in figure 3.

An experiment using 50 and 100 generated input files and three workstations, one master/slave and two slaves were conducted. All three workstations had the same processor type, same amount of RAM, and same available hardware. Slave registration was also based purely on processor utilization and registration was only accepted if a slave's processor utilization was below 60%. In this large experiment, the environment was not controlled and the workstations were occupied by users. The results of this experiment are shown in figure 5. Since the workstations were homogeneous, one computer was used to provide the sequential execution time. One computer took 29.2 minutes to execute 50 simulations and 72.27 minutes to perform 100 simulations. With TREx, three workstations took 16 minutes for 50 simulations and 32.7 minutes for 100 simulations. TREx took 54.8% and 45.2% of the sequential time for 50 and 100 simulations respectively. Obviously, when the number of slaves is equivalent to the number of simulations being launched, then the results would be most optimal. However, just by using three existing workstations within their network, speedup improved by a factor of two.

The previous experiment used TREx to run multiple simulations for Monte Carlo analysis. Certain parts of the Odyssee simulator were modularized and TREx was used to gain speedup improvements. Persee, a pre-processing component within Odyssee, was modularized in order for concurrent computing to be possible. The Persee component defines the simulation zones covered for each system (2G, 3G, WLAN, WiMAX) and the overlap of the pre-defined base stations for each system. The modularization of Persee is shown in figure 6. There are three main steps within Persee:

1. Load Clutters: Load all systems' base station information
2. CalculerTabCoords(): Define useful areas covered by the systems' base stations within the entire simulation zone.
3. CalculerTabMailles(): Construct a data structure containing information on all the useful areas.

Defining the simulation zones is independent for each system. Therefore, computing the base station information and simulation zone for each system can be done easily with minimal code modifications. Each system module sends data to the "main module" where the data is synchronized and the remainder of Persee is executed normally.

Since large simulations were conducted, TREx was used with France Telecom's servers that can support the large environments. Two environments were used in the experiment. The first contained 701 2G and 666 3G base stations. The second contained 701 2G, 666 UMTS, 666 WiMAX, and 666 WLAN base stations. Since the servers are not homogeneous, individual Persee running times for the first environment on each server are shown in figure 7 as well as the performance of Persee for two servers, p-sein and p-cigale, using TREx.

Since TREx parallelizes Persee for each system and there are only two systems for the first environment, the level of parallelization is low. The fastest server, p-djerba takes 249 seconds and the TREx federation takes 215 seconds. Note that p-djerba is approximately 20% faster than the average

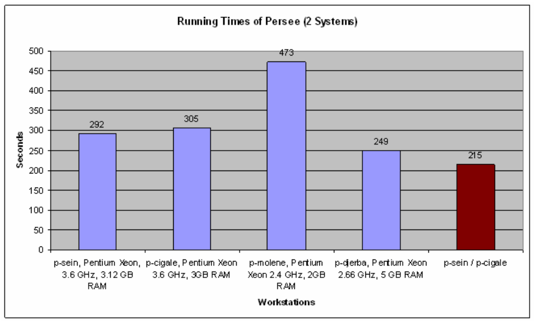


Figure 7: Running Times of Persee with 2 Systems

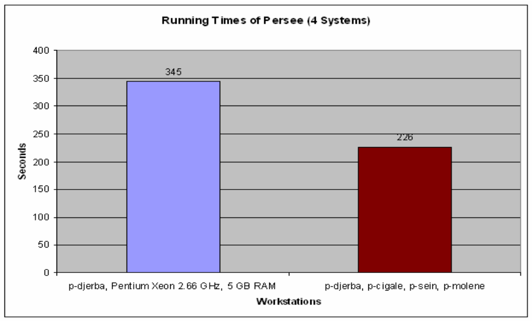


Figure 8: Running Times of Persee with 4 Systems

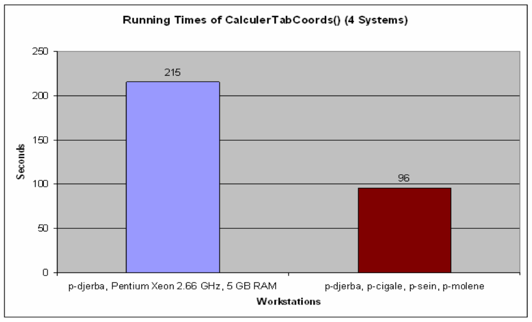


Figure 9: Running Times of CalculerTabCoords() with 4 Systems

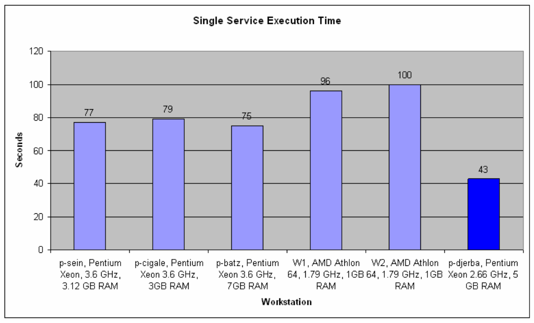


Figure 10: Running Times of a Single Expodys Service

of the two servers used in TREx. However, the using TREx federation takes 86% of the running time of p-djerba.

Figure 8 shows the running time of Persee with the second environment using p-djerba and TREx using a p-djerba, p-cigale, p-sein, and p-molene. P-djerba takes 345 seconds and the TREx federation takes 226 seconds to perform Persee. The TREx federation takes 65% of the execution time for p-djerba.

Figure 9 shows a closer look at the CalculerTabCoords function being parallelized. The times recorded represent the start of the simulation to the start of CalculerTabMailles function. This includes the writing, reading, and transferring of data for the TREx federation. P-djerba took 215 seconds to finish CalculerTabCoords while TREx took 96 seconds for the main module to obtain and synchronize the data from all the modules. Looking at this section of Persee, TREx took 44% of the execution time of p-djerba.

Another part of Odyssee, Expodys, can be parallelized using TREx. Expodys is a post-processing component within Odyssee that creates QoS maps for different types of mobile services (voice, visio, ftp, web browsing, etc.). Each service that Expodys computes the maps for are independent and defined in the Odyssee input files. Therefore, distribution and concurrent computing using TREx can easily be done without any code modifications. A single service is defined in the input file and the TREx slaves will compute the data for this service and send it back to the TREx master.

Figure 10 shows the execution times on a hybrid of workstations and servers used in the experiments. The fastest workstation, p-djerba, executed the single service in 43 seconds while the slowest workstation, W2, executed the single service in 100 seconds.

Figure 11 shows the execution times for 5, 10, and 20 Expodys services executed on p-djerba and a TREx federation containing workstations p-sein, p-cigale, p-batz, W1 and W2. The TREx federation took 54.6%, 44.2%, and 30.8% of the execution time of p-djerba with 5, 10, and 20 services respectively.

Besides Odyssee, TREx can be used for other applications. TREx was used to distribute SciLab computations over a cluster of five heterogeneous occupied workstations. The SciLab computations were used to produce results computed from an algorithm that used random variables for Monte Carlo analysis. The processing capability for this cluster consisted of three Pentium 4's, one AMD Athlon, and one Pentium Duo Core. Table 2 workstation to compute 20 SciLab computations and the time it took TREx cluster to perform 20 SciLab computations.

A single SciLab computation takes 42 minutes and 30 seconds on the fastest workstation in the cluster. It is projected that it will take 14 hours and 9 minutes to complete 20 SciLab computations at this rate. The TREx cluster completed 20 SciLab computations in 4 hours and 36 minutes. With five workstations, TREx has produced speedup by approximately a factor of three.

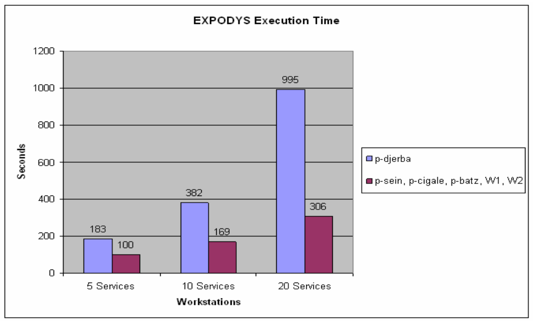


Figure 11: Running Times of Expodys with 5, 10, and 20 Services

	Single Computation	20 Computations
Pentium Duo Core Processor	42 min 30 sec	14 hours 9 min
TREx Cluster	-	4 hours 36 min

Table 2: SciLab Results using TREx and a Single Workstation

6. Future Work and Conclusions

The objective of TREx is to provide a low-cost solution for scientific computing in institutional networks. Odyssee parametric experiments achieved more than a 50% speedup improvement using only three workstations. Modularization of Odyssee to use TREx has significantly improved Odyssee's pre-processing and post-processing execution time. Distributed SciLab computations gained a speedup factor of three using five workstations. These large improvements were achieved at no cost since the communication infrastructure already existed and no additional hardware needed to be purchased.

TREx is unique since resource management is fully distributed, slave registration is dependent on the variability of its resources, and masters have control of assigning services to the slaves. This distributed resource management provides a "fair" allocation of slaves per master and also provides a simple and dynamic way to see if a slave can be used for remote processing.

Ways to dynamically federate clusters opens an important field of research. Currently, a network administrator can intervene and re-configure the clusters, but an optimal solution would be for the federations to dynamically configure themselves. Optimal resource management and scheduling algorithms with this architecture are still in the research and development stages.

Current and up-coming applications can be designed using this parallelization framework. Additional modifications of Odyssee to fit the TREx parallelization framework are being conducted.

Performance results between a sequential machine and the TREx system will be evaluated using different scenarios within Odyssee and other applications. Results of a high performance distributed computer can be obtained at a minimal cost.

References

[1] R. M. Badia, J. Labarta, R. Sirvent, J. M. Pérez, J. M. Cela, and R. Grima, Programming Grid Applications with GRID Superscalar, Journal of Grid Computing, January 2004.

[2] T. Brecht, H. Sandhu, M. Shan, and J. Talbot, ParaWeb: Towards World-Wide Supercomputing, Proceedings of the Seventh Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, September, 1996.

[3] A. Bricker, M. Litzkow, M. Livny. Condor Technical Report, TR 1069. CS dept., Univ. of Wisconsin-Madison, Jan 1992.

[4] J. Chase , F. Amador , E. Lazowska , H. Levy , and R. Littlefield, The Amber System: Parallel Programming on a Network of Multiprocessors, Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, pp.147-158, November, 1989.

[5] R. Davoli, L.A. Giachini, Ö. Babaoglu, A. Amoroso, and L. Alvisi, Parallel Computing in Networks of Workstations with Paralex, IEEE Transactions Parallel and Distributed Systems, vol. 7, no. 4, pp. 371- 384, April 1996.

[6] S.C. Kendall, J. Waldo, A. Wollrath and G. Wyant, A Note on Distributed Computing, Sun Microsystems Technical Report TR-94-29, Sun Microsystems, CA, November 1994

[7] J. Linderoth, S. Kilkarni, J.-P. Goux and M. Yoder, An Enabling Framework for Master-Worker Applications on the Computational Grid, Proceedings of the Ninth IEEE Symposium High Performance Distributed Computing, pp. 43-50, August, 2000.

[8] M. Litzkow, M. Livny, and M. Mutka, Condor - A Hunter of Idle Workstations, Proceedings of the Eighth International Conference of Distributed Computing Systems, pp. 104-111, 1988.

[9] A. E. Samhat, Z. Altman, M. Francisco, and B. Fourestie, Semi-dynamic Simulator for Large-scale Heterogeneous Wireless Networks, International Journal of Mobile Network Design and Innovation, vol. 1, no. 3-4, pp 269-278, 2006.

[10] A. S. Tanenbaum and M. van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, Upper Saddle River, New Jersey 07458, 2002.

[11] S. Vinoski, Distributed Object Computing with CORBA, C++ Report, Vol. 5, July/ August 1993.

[12] J. Waldo, The Jini Architecture for Network-Centric Computing. Communications of the ACM, 42(7):76–82, July 1999.