# The Distributed Clearance Project:

## *A Case Study in Distributed Computing*

**James E. Ries[1,2,3], M.S.**
**Gordon K. Springer[1], Ph.D.**

**[1]Department of Computer Engineering and Computer Science**
**[2]Department of Health Management and Informatics**
**University of Missouri**
**Columbia, MO 65211**

**[3]Engineering Animations, Inc.**
**1000 W. Nifong Blvd. Bldg. 1**
**Columbia, MO 65203**

**ABSTRACT**

Distributed computing promises to break through computational performance barriers by cutting large problems into small pieces which can be solved by many machines acting in concert. However, not all problems are good candidates for this sort of "divide and conquer" strategy. Even large, compute-intensive problems that appear easily divisible on the surface may have dependencies on data and dependencies across partitioned units of work that may prove to negate the advantages associated with distributing the workload.

The Distributed Clearance Project demonstrates both the potential and the pitfalls of distributed computing. The project seeks to take a very computationally intensive task (calculating part clearance tolerances in large manufacturing models) and break it into jobs which can be performed independently by diverse and geographically remote computing resources. The project relies on Distributed Computing Environment (DCE) standard Remote Procedure Calls (RPC) as a platform-independent transport mechanism, allowing a heterogeneous mix of computers to participate.

Though the atomic transactions (part clearance calculations) that form the basis for the Distributed Clearance Project are essentially independent of each other, some of the existing non-distributed code on which the project is based contains assumptions regarding relationships among these transactions. Since the project was undertaken with a goal of reusing existing non-distributed code, it was not until initial evaluation of the project that these hidden dependencies became apparent. In fact, the assumptions made in the non-distributed code significantly degrade performance of the distributed solution.

This paper compares the performance of a non-distributed solution with the performance of the Distributed Clearance Project. We discuss the hidden assumptions which affect the Distributed Clearance Project and changes that are being made to improve the partitioning of the problem space so more nearly optimal distributed performance can be achieved. We also point out "lessons learned" in building distributed solutions from existing non-distributed code.

**Keywords:** Distributed Computing, Load Balancing, Transactions, Performance, Partitioning.

**INTRODUCTION**

Many scientific papers which address load balancing or distributed computing in general assume that the domain of such problems is really supercomputing or high-end cluster computing. This assumption is rapidly becoming erroneous. Distributed computing is moving from high-end (and high cost) computing into the mainstream. Distributed computing is now being employed in industry to make computationally intensive problems not only tractable, but also economically feasible. We call distributed computing that operates using standard workstations, networks, and operating systems, "commodity" distributed computing.

The Distributed Clearance Project is an effort by Engineering Animations, Inc. (EAI; see www.eai.com) to take an existing processor-intensive task, break it into independent sub-tasks, and distribute these sub-tasks across a generic local area network to a group of heterogeneous computers. EAI's products support a variety of Unix flavors as well as Microsoft operating systems on the Intel platform. Like most commercial projects, part of the initial goal was to make use of as much existing code as possible.

The project was somewhat inspired by the success of the SETI @ Home project [1]. As we designed the Distributed Clearance Project, it became apparent that our project had some essential features in common with SETI @ Home that made it a good candidate for a similar sort of gross strategy in distributing units of work. SETI works because the units of work performed by each of the servers are quite time-intensive. This characteristic is critical in commodity distributed computing because the time to make a call or perform a transaction can be relatively high as compared to Message Passing Interface (MPI) or other schemes typically employed in high performance distributed computing environments. In the

case of SETI, units of work typically take many hours or even days to complete, so the time needed to communicate between the client and server is largely irrelevant. As we will show, communication time is somewhat relevant for the Distributed Clearance Project.

Another important aspect of SETI is the independence of each unit of work. For SETI, each unit of work is completely independent of all others. Again, this is only partially true for Distributed Clearance.

## THE PROBLEM

Many of EAI's software products deal with large manufacturing data models. For example, a typical customer might be using EAI products to analyze the design of a motor vehicle. One type of analysis that is frequently of interest is interference and clearance. This analysis reports parts of a model that overlap or penetrate each other (interference) and/or parts that are closer to each other than a given tolerance (clearance).

The interference/clearance calculations must be performed many times throughout the manufacturer's design process as changes occur in specifications, part designs, module designs, etc. In large models, these calculations can be extremely time-consuming. Since each part must be "cleared" against every other in the model, approximately $n^2$-n calculations may be performed (the -n term comes from the diagonal of a matrix since it is plainly unnecessary to clear a part against itself).

The individual clearance tests are normally performed on "tessellated" renditions of the parts using computational geometry methods. A tessellation is a transformation which represents the part as a number of simple polygons. For more demanding needs, exact or Non-Uniform Rational B-Splines (NURBS) [2] data can also be used, but at the price of additional computation time.

As mentioned, many EAI customers work with extremely large models containing perhaps hundreds of thousands of parts. Some early pruning can be employed to eliminate parts which clearly are not of interest (e.g., parts on opposite sides of a very large model may be obviously far enough apart to obviate the need for a more precise calculation). However, it still may be necessary to perform literally billions of clearance computations. Even on relatively fast workstations, the entire process can take days to complete.

## THE SOLUTION

It is extremely inconvenient for customers to wait days for important interference/clearance results. In order to improve this time, additional processing power must be brought to bear. Fortunately, the clearance calculations for individual parts are largely independent, and so it was felt that processing many part clearances in parallel could greatly speed the overall procedure.

Although EAI products are conceptually based on Microsoft's component Object Model (COM) [3], EAI also supports a large number of Unix platforms. A given customer may be running EAI products on several different platforms, and it is important that all inter-operate as seamlessly as possible. In order to address this need for cross-platform interoperability, we designed a custom mechanism for distributing clearance work to available servers. We considered available off-the-shelf solutions such as Microsoft Transaction Server (MTS) [4] or IBM's LoadLeveler [5]. Though these commercial solutions contain many features that our own solution lacks, they are somewhat bound to specific platforms and thus unsuitable for our needs.

Essentially, our solution uses standard DCE RPC's to send jobs (to calculate the clearance of two parts) to a central server, which we call the Queue Server. The Queue Server forwards these jobs to a variety of Work Servers (servers capable of processing the jobs). By using DCE RPC's, we can assure native operating system support on virtually all major Unix platforms with the notable exception of Sun. However, even the Sun platform can be supported with low cost third-party software. It is noteworthy that we rely only upon the RPC portion of DCE, as additional DCE functionality such as the Cell Directory Service (CDS) cannot be guaranteed to be available on all platforms [6].

Work Servers register with the Queue Server and then request jobs from the Queue Server as needed. Since Work Servers only request an additional job when they have completed their previous work, fast servers tend to receive more work than slow servers do. When a Work Server completes a job, it calls back the Queue Server with the result and the Queue Server then matches this result with the appropriate client and forwards the callback to the client.

On the other side of the communication, clients register with the Queue Server and send potentially many jobs to the Queue Server. Clients act as a DCE server by listening for callbacks that detail the results of a particular job. The Queue Server has separate threads of execution to deal with listening for client requests, sending jobs to Work Servers, listening for results, and returning results to clients. Since the Queue Server will likely hand out a given client's job requests to many different servers, the results come back asynchronously to the client via the callback interface.

Figure 1 shows the general architecture of this distribution scheme. The darker arrows depict the call/callback mechanism used to move clearance requests and results respectively from clients to the Queue Server to Work Servers and back through the chain. The lighter arrows sketch the underlying control mechanisms that are used to identify Clients and Work Servers to the Queue Server, thus allowing the Queue Server to manage the overall communication flow. While the diagram shows the interaction of one Client with one Queue Server and one Work Server, the ellipses (…) indicate that in practice there would certainly by many Work Servers. The connections between the Queue Server and additional

Work Servers have been omitted for clarity. There also could be many clients simultaneously operating as well.

Figure 1 contains numbered arrows to indicate the order of calls in the architecture. This is not strictly accurate in that the architecture is asynchronous in many respects. However, these numbers provide a sample scenario that illustrates a typical interaction. A Client registers with the Queue Server on its ServerControl interface (1). This registration call provides information necessary for later callbacks. The client may then register on the QueueControl interface (2), which provides special management information to the Queue Server.

Independently, the Work Server registers with the QueueControl interface (3) providing information necessary for the Queue Server to locate it in the future. The Queue Server responds by registering as a client of the Work Server on the ServerControl interface (4).

Finally, work proceeds by the Client issuing a custom call (e.g., a clearance calculation request) to the Queue Server (5). When appropriate (based on available Work Servers, load of the Queue Server, etc.), the Queue Server sends the same custom call to one of the registered Work Servers (6). When the work has been completed, the Work Server calls back to the Queue Server with the result (7) and the Queue Server, in turn, calls back to the Client with the result (8).

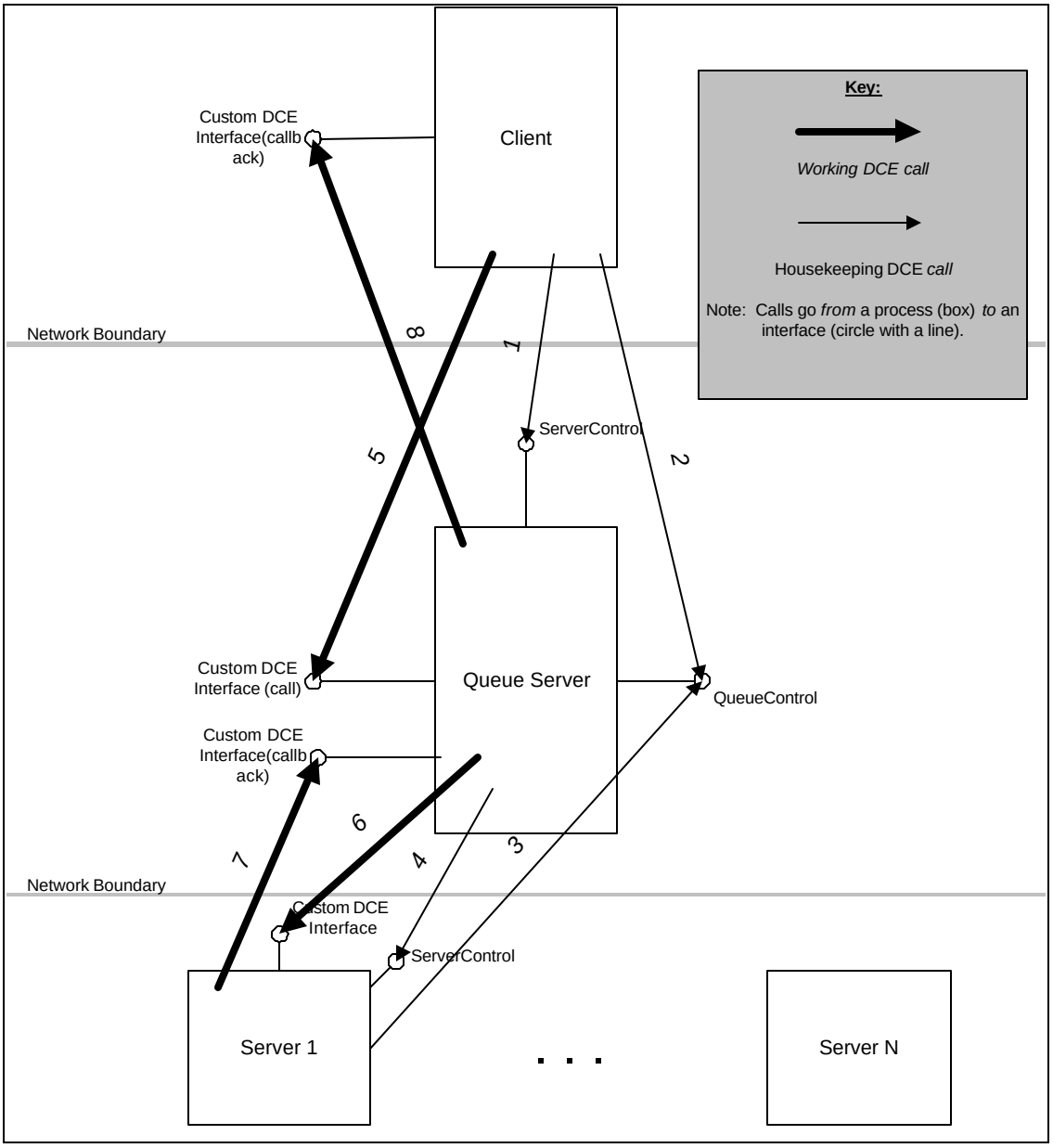


**Figure 1: Distributed Clearance Architecture**

## PERFORMANCE RESULTS

Table 1 displays a number of timings of the standard (non-distributed) clearance module and of the distributed clearance module, which we developed. The first and fourth lines characterize the old non-distributed calculation. All other lines were timed using the distributed clearance module.

Notice that the first three lines of result are concerned with "local data". Typically, the model data involved in clearance calculations is stored in a shared file system (Windows NT File Sharing in our test, but AFS, DFS, and the like are also common). These first three lines of result show that copying data to the local file system of the server dramatically affects performance and the ability of the system to scale. All other tests were done with the model data contained on a shared file system.

All test machines involved in the results presented in Table 1 were 400 MHz Pentium III machines running either Windows NT or Windows 98. The system was also tested using an HP/UX machine for proof of concept, but rigorous timing data were not kept.

| Local/Remote | Protocol | Count Servers | Exec. Time (secs) | Exec. Time per Server (secs) | Calcs | Calcs per sec. | Msecs per calc | Improvement | Specific Machines Used | Server-reported Msecs per calc | RPC Overhead |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Local - Local Data | N/A | 1 | 121.81 | 121.81 | 1493 | 12.26 | 81.58 | 0.00% | 1 | 81.58 | 0.00% |
| Remote - Local Data | LRPC | 1 | 158.56 | 158.56 | 1493 | 9.42 | 106.20 | -30.17% | 1 | 106.19 | 0.01% |
| Remote - Local Data | TCP | 5 | 29.85 | 5.97 | 1493 | 50.01 | 20.00 | 75.49% | 1,2,3,5,6 | 18.19 | 9.95% |
| Local | N/A | 1 | 284.93 | 284.93 | 1493 | 5.24 | 190.84 | 0.00% | 1 | 190.84 | 0.00% |
| Remote | TCP | 1 | 327.88 | 327.88 | 1493 | 4.55 | 219.61 | -15.07% | 1 | 206.23 | 6.49% |
| Remote | TCP | 1 | 451.30 | 451.30 | 1493 | 3.31 | 302.28 | -58.39% | 2 | 289.74 | 4.33% |
| Remote | TCP | 1 | 270.11 | 270.11 | 1493 | 5.53 | 180.92 | 5.20% | 3 | 157.68 | 14.74% |
| Remote | TCP | 1 | 429.07 | 429.07 | 1493 | 3.48 | 287.39 | -50.59% | 4 | 274.90 | 4.54% |
| Remote | TCP | 1 | 297.84 | 297.84 | 1493 | 5.01 | 199.49 | -4.53% | 5 | 187.73 | 6.26% |
| Remote | TCP | 1 | 295.27 | 295.27 | 1493 | 5.06 | 197.77 | -3.63% | 6 | 186.11 | 6.27% |
| Remote | TCP | 2 | 272.33 | 136.17 | 1493 | 5.48 | 182.41 | 4.42% | 3,6 | 170.62 | 6.91% |
| Remote | TCP | 3 | 213.37 | 71.12 | 1493 | 7.00 | 142.91 | 25.12% | 3,5,6 | 131.39 | 8.77% |
| Remote | TCP | 4 | 219.12 | 54.78 | 1493 | 6.81 | 146.76 | 23.10% | 1,3,5,6 | 134.77 | 8.90% |
| Remote | TCP | 5 | 219.90 | 43.98 | 1493 | 6.79 | 147.28 | 22.82% | 1,2,3,5,6 | 131.11 | 12.34% |

**Table 1 : Performance Results**

## DISCUSSION

Clearly, the results were not tremendously encouraging. Although there was a fair amount of variance from one trial to another, it seems clear that a performance "ceiling" exists at about three servers. In addition, it is quite troubling that the RPC overhead (the percentage of time spent in communication rather than actual computation) seems to grow significantly as the number of servers increase.

As we looked at these numbers a bit more and did some ancillary tests, it became apparent that several factors were responsible for our lackluster performance. First, a tremendous amount of time is being spent finding and loading data from the large model files. Our Distributed Clearance Project actually makes this worse, since a number of servers may have to search for a given part. The non-distributed version may find this part once in the file and utilize the data n-1 times when clearing it against other parts. This problem is quite similar to cache coherency problems in multi-processor and cluster computing environments [7], [8].

Secondly, the local area network on which we conducted the tests is not sufficient. The network we used is a 10 megabit Ethernet and it is fairly loaded with other activities at most times. We tried to conduct many of the tests during off-hours to minimize the effects of other network traffic. However, timed jobs may have interfered somewhat with some of the tests. More to the point, we found that as the number of servers increased and began to rapidly attempt to access the model files, collisions on the Ethernet increased dramatically. Thus, as the number of servers concurrently trying to load part data increases, the Ethernet collisions effectively put a cap on throughput. Further evidence of the collision effect is the RPC overhead, which of course increases as collisions increase.

## FUTURE WORK

We are currently working to generalize our job distribution scheme so it can be used with types of work besides interference/clearance. We call the generalized architecture, the Transaction Server Architecture (TSA). TSA utilizes Microsoft COM principles to allow pluggable components to abstract away the details of a

specific job type. For example, the Transaction Server (formerly called the Queue Server) requires a "marshalling component" for every job type it supports. Thus, it can be extended to support any new job type, by simply implementing a new marshalling component.

In addition to generalization, we are of course very interested in improving performance. In order to maintain our new found generality, we have added the concept of a "selection component." A selection component allows a Transaction Server to choose a job bound for a specific server from among available jobs. In the base case, this may be simply taking the first job out of the data structure (as was done in the old Queue Server). However, in cases where careful job selection can improve performance, other algorithms can be incorporated. For example, in the case of distributed clearance, it should certainly improve performance to give a server a part that it has worked on in the recent past (since the part may still be in cache).

We also hope to improve performance through another modification. As we have shown, the original Distributed Clearance Project simply distributed each individual clearance calculation. In TSA, we anticipate the capability to provide groups of calculations to a given Work Server. The danger of this approach is that a slow server may receive a large number of calculations and may become the bottleneck to the entire computation. However, we believe in practice this will merely be a matter of tuning. We could introduce a more complex scheme to negotiate an appropriate number of transactions for a given machine based on its capabilities. However, it is felt that this additional complexity would not be worth the runtime overhead and design-time programming hours it would require, considering our purposes.

One could mitigate the need to send transactions involving a given part to the same Work Server by sending all part data along with the transaction. This would prevent multiple Work Servers from repeatedly seeking through the part database for the same data. However, performance could still be improved in this case through the above selection strategy and grouping strategy. In addition, this scheme does not lend itself well to the use of current calculation code, and reusing this existing code is highly desirable.

Although DCE RPC is generally a good choice for transport, there may be situations in which this protocol is not ideal. For example, in a business-to-business setting, DCE RPC is often cited as being problematic since it does not lend itself well to crossing firewalls. To this end, we are considering development of a pluggable transport layer for TSA. Initially, we would implement a component that simply provides DCE RPC, but additional components for specific environments could follow (e.g., Simple Object Access Protocol or "SOAP" [9] for business-to-business applications).

## LESSONS LEARNED

Although some of the lessons learned in the Distributed Clearance Project are certainly specific to the given environment, it is felt that some things generalize. The remainder of this section summarizes the lessons taken away from Distributed Clearance.

*Units of work should be independent in every way if possible*. Although given operations may be functionally independent, consider their relationships with regard to performance. In the Distributed Clearance Project, each of the transactions was independent, or did not rely on results of the others. However, the fact that a part involved in one of the transactions could be used repeatedly greatly improved the performance of the non-distributed case, while a naïve distribution mechanism defeated this.

*Time to process a unit of work should ideally be several orders of magnitude greater than the time to send a request across the network*. To truly scale up well, a distributed application cannot spend a large fraction of its time sending packets back and forth across the network. If possible, try to batch up larger chunks of work into a transaction to lessen the effect of network latency.

*Analyze an application's use of the network before attempting to turn it into a distributed program*. The non-distributed version of the clearance calculation module did not explicitly use the network. However, its heavy reliance on shared file system files greatly affected the performance of the distributed implementation.

## CONCLUSION

Distributed computing remains somewhat of an art. Though advances have been made in frameworks, compilers, distributed operating systems, etc., implementation of a given distributed system still often requires a great deal of effort. It is still extremely important to apply careful design, and to test assumptions in order to achieve good performance, perhaps more so in distributed computing than in other areas.

Additionally, commodity distributed computing still sits atop a scare resource: network bandwidth. Although our test cases were severely limited by the 10-Mb bandwidth of our Ethernet, even "fast" 100-Mb networks are not sufficient for many applications. Researchers must recognize that the gigabit and switched networks in research labs will remain unreachable for many business applications for some time to come.

Finally, some applications are good candidates to be turned into distributed applications and some are not. This is not always readily apparent. Applications such as SETI @ Home can afford to wait weeks or months for results as long as those results eventually arrive, and so distributing the application's transactions is trivial. For more common applications where both overall and individual performance is paramount (such as the Distributed Clearance Project), the distribution algorithm can be critical. Careful analysis and testing are required to determine the potential for breaking a non-distributed application into parallel units that can be distributed.

**ACKNOWLEDGEMENTS**

This work was done under a contract for Engineering Animations, Inc. of Ames, IA. Mr. Ries is also supported by the National Library of Medicine, grant LM-07089-08. The authors would like to thank Jerry Spratt of Engineering Animations, Inc. for his many contributions to the design of the project, and for his excellent implementation of the Queue Server.

**REFERENCES**

[1] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Anderson, "A new major SETI project based on Project Serendip data and 100,000 personal computers", in Proceedings of the Fith Intl. Conf. on Bioastronomy, 1997. Also available at: http://setiathome.berkeley.edu/woody_paper.html.

[2] L. Piegl, W. Tiller, The NURBS Book, Second Edition, Springer-Verlag, 1996.

[3] Microsoft Corporation, "Component Object Model Home Page", available at http://www.microsoft.com/com/default.asp.

[4] Microsoft Corporation, "Microsoft Transaction Server Home Page", available at http://www.microsoft.com/com/tech/mts.asp.

[5] IBM Corporation, "Loadleveler Home Page", October 1998, available at: http://www.austin.ibm.com/software/sp_products/loadlev.html.

[6] J. Ries, "Interconnecting Personal Computers with the Distributed Computing Environment", University of Missouri Thesis, 1998.

[7] R. Giorgi, C. A. Prete, "PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors", in IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 7, July 1999.

[8] M. M. Michael, A. K. Nanda, B. Lim, "Coherence Controller Architectures for Scalable Shared-Memory Multiprocessors", in IEEE Transactions on Computers, Vol. 48, No. 2, February 1999.

[9] D. Box, G. Kakivaya, A. Layman, S. Thatte, D. Winer, "SOAP: Simple Object Access Protocol", IETF Internet Draft, November 1999, available at: http://search.ietf.org/internet-drafts/draft-box-http-soap-01.txt.