2008-04-02

Increasing DOGMA Scaling Through Clustering

Nathan Hyrum Ekstrom
Brigham Young University - Provo

INCREASING DOGMA SCALING THROUGH CLUSTERING

by

Nathan Ekstrom

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Nathan Ekstrom

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Quinn Snell, Chair
Date	Mark Clement
Date	Kevin Seppi

ABSTRACT

INCREASING DOGMA SCALING THROUGH CLUSTERING

Nathan Ekstrom

Department of Computer Science

Master of Science

DOGMA is a distributed computing architecture developed at Brigham Young University. It makes use of idle computers to provide additional computing resources to applications, similar to Seti@home. DOGMA's ability to scale to large numbers of computers is hindered by its strict client-server architecture. Recent research with DOGMA has shown that introducing localized peer-to-peer downloading abilities enhances DOGMA's performance while reducing the amount of network and server usage[1]. This thesis proposes to further extend the peer-to-peer abilities of DOGMA to include peering client server communication by creating dynamic clusters of clients. The client clusters aggregate their communication with only one client communicating with the server directly. This further reduces the network traffic and server usage allowing more clients to connect to a single server and increasing the overall scalability of DOGMA systems.

ACKNOWLEDGMENTS

I would like to thank my wife Jessica for her unwavering support and never letting me give up, my parents for always pushing me to do my best, and my committee Dr. Snell, Dr. Clement, and Dr. Seppi for their patience with a part time student.

Contents

Chapter 1

Introduction and Motivation

1.1 Introduction

Distributed computing is used for MIMD applications which can be run on multiple machines across a network. It can involve thousands of machines or only a few and is an important research tool in many areas. Recent years have seen an increase in both the use and availability of distributed computing systems: Boinc (Seti@home)[2], Folding@home[3], XGrid[4], Condor[5], and Globus[6] are some examples. All of these systems allow researchers to split a large task into smaller parts for processing on numerous computers. An important facet of many of these systems is that they allow idle machines to be put to productive use.

At Brigham Young University another system for distributed computing has been developed called DOGMA(Distributed Object Group Meta Computing Architecture). It has been used for tasks such as distributed rendering of computer graphics and phylogenetic tree search.

1.2 Motivation

DOGMA uses a client server architecture, making the server a bottleneck when large numbers of clients are available for use. Previous research[1] has shown that by introducing peer-to-peer techniques into DOGMA, specifically for downloading data used for processing, significant performance increases occur. These performance increases allow for a less powerful server, less bandwidth between clients and server, more clients connecting to a single server, and lower operating costs. These increases make the system more available to researchers with a limited budget.

DOGMA still has scaling issues related to client server communication. As more clients join the system the server becomes bogged down and unable to handle the load. The amount of network traffic to and from the server as well as active connections grows linearly with the number of clients and can appear to network monitoring systems as a denial of service attack. An effective distributed computing platform should be able to add more clients without a linear increase in network and processor usage at the server.

1.3 Hypothesis

DOGMA is unable to scale to large numbers of clients because of the bottleneck created by server resources. Introducing peer-to-peer technologies to DOGMA's communication layer will allow clients to cluster together and have a single member aggregate all client server communication for the group. Clustering will reduce the network and server load, allowing the system to scale to larger numbers of clients.

Chapter 2

Related Work

2.1 Local Client Server Distributed Application Systems

Condor[5], Globus[6], and Legion[7] are similar systems for distributed computing. They all use a client server architecture where the servers are *near* the clients, i.e. do not cross organizational boundaries. Both Condor and Globus provide a mechanism for scaling while Legion does not explicitly provide any.

Condor uses an idea termed ”flocking” which allows multiple groups or ”flocks” to share work providing a larger pool from which to draw resources. Flocking is also their method of allowing clients to be used across organizational boundaries. Each organization or group of clients has their own scheduling server. Each scheduler then is configured to communicate with the other servers when resources are needed or available.

Globus provides a similar mechanism where servers can setup communication with each other and share resources. Globus also provides a mechanism for creating a hierarchy of servers. Both of these systems are effective but they require a great deal to setup, maintain, and run. They are also unable to scale dynamically with load and instead require static setup of additional servers.

2.2 Internet Wide Distributed Application Systems

Boinc(Seti@Home)[2], and Folding@Home[3] are two well known systems which allow people to donate their idle processor time to a cause. Each uses a client server architecture. A major difference between DOGMA and these systems is client management. Both of these systems give a large chunk of data to a client and then, in essence, forget about them until the results are returned. To ensure a result for a given data set the same set is given to multiple clients. This causes wasted processing time when more than one client finishes with that set. DOGMA is more real-time; wanting frequent updates so that it can reassign work if a client goes off-line.

2.3 Java Distributed Computing Library

The Java Distributed Computing Library(JDCL)[8] is the distributed computing platform most similar to DOGMA which uses a client-server architecture written in Java. It is not as flexible as DOGMA and is meant to only run programs which use the JDCL.[9]

Similar to DOGMA the JDCL suffers from problems of scaling[10]. To alleviate this problem they create a multi-tiered server architecture to make their system "scale-free". Their approach creates a n-tiered tree of nodes. Internal nodes, which act as servers, receive work from their parent and try to balance that work among all of their children. This continues until it reaches a leaf node where it is finally processed. Their hierarchy of server/schedulers is effective at allowing their system to scale. However it has several issues which make it inappropriate for implementation in DOGMA. First, they require additional dedicated servers to improve their scaling which introduces administrative overhead. Second, each server in the hierarchy has to be setup and configured by hand which can be difficult and sometimes wasteful. For example, if many nodes join the tree and additional servers were configured to accommodate them, when the nodes leave, the servers are still running until they are manually removed. Third, JDCL programs either are not able to take advantage of locality or they need the ability to setup a server node in the same area as the nodes. This can be difficult to do when crossing organizational boundaries.

2.4 Proxies

Proxies act as mirrors or relays for clients. A proxy acting simply as a mirror is inappropriate for DOGMA because all requests generate dynamic responses. Proxies also require manual setup and introduce organizational management issues.

2.5 MapReduce

MapReduce is a model of programming developed at Google[11]. It uses a map function which takes a key/value pair as input and returns a set of intermediate key/value pairs. The intermediate key/value pairs with the same key are all given to a reduce function which merges them together providing a new list of key/value pairs. Many problems can be expressed in this way but it is not a general purpose distributed computing model like DOGMA.

An important fact of Google's implementation of this model is their use of locality. They have extremely larges sets of data to process and network bandwidth is at a premium. As a result they try to schedule work at nodes that already have the desired data or at ones near nodes with the needed data.

2.6 Peer-to-peer

Peer-to-peer systems use direct peer communication allowing for the elimination of the server. These systems can scale extremely well. However, they lack a central point of control and entry. A complete peer-to-peer system is inappropriate for our situation and would make the system much more difficult to manage. Adding aspects of peer-to-peer systems however, has already been shown to increase the scalability of DOGMA[1].

Recent research has shown that incorporating peer-to-peer technology into the file distribution has increased its ability to scale by reducing network and server usage. The research introduces LURP (Local URL Resolution Protocol)[1]. Clients using LURP ask local neighbors if they already have a file. If a neighbor has it, the client gets the file from the neighbor. In the case where a neighbor does not have the file but is currently downloading it, the client waits for the neighbor to finish and then acquires the file from it. This use of locality allows DOGMA clients to be less demanding on limited network resources.

2.7 Summary

Current distributed computing systems all have one of two problems. Either they require additional server infrastructure to handle more clients or they cannot handle general computing tasks, requiring the use of their library. DOGMA has the first problem, it requires a more powerful server to continue to scale. Adding peer-to-peer techniques into client server communication will reduce the need for a more powerful server while still increasing scaling and performance.

Chapter 3

DOGMA

3.1 Introduction

DOGMA is a distributed computing scheduler designed to work in a shared nothing environment and processing elements may come and go at any time. It works best with trivially parallel jobs which can be easily divided into specific parts.

A job consists of one or more "app-parts". An app-part is a piece of work which can be given to an individual computer to complete. In order to successfully run an app-part, the machine running it requires a command line to execute as well as any files needed for the job.

Different client machines may be running different operating systems with different CPU architectures, for example x86 or PPC, so DOGMA has developed the notion of a platform. A platform is simply an operating system and CPU architecture combination. Command lines are associated with platforms so that when the server is giving clients app-parts to work on it can give a command line that will work for that specific machine.

DOGMA also has the notion of commands. Commands are just message types which tell a client to start or stop working on an app-part. In the future commands could be added for pausing, resuming, connecting to a specific client cluster, etc.

3.2 Client

There are two separate pieces which make up the client. The first piece is the bootstrapping mechanism. The bootstrapper is a simple piece of code which takes a URL to a Java class repository and a class to run. It then loads the repository like a jar file and executes the *Run* function on that class. The system could launch the DOGMA client directly with this mechanism. However, this would have the undesired consequence of greatly increasing network traffic because any time the client loaded a class it would force a network call. Instead, the bootstrapper loads a class which downloads the latest DOGMA jar files and dependencies putting them in their own separate class loader, this prevents network calls from being made when classes are loaded. It then executes the *Run* function on the main DOGMA client, whose name is acquired from a dynamically generated properties file provided by the server. The boostrapping mechanism could be used to launch almost any Java based application.

The main DOGMA client performs some initial setup of directories and auxiliary services and then enters its main loop. The main loop does only two things. First it does a check in with the server providing information about what the client is currently working on. It then executes any commands it receives back. Currently only start and stop commands exist. If the client receives back no commands it continues what it is doing.

3.3 Server

The DOGMA server consists of three main parts. They are the communication layer, the management website, and the scheduler. The communication layer makes use of the SOAP protocol and is responsible for handling communication with clients as well as providing a programmatic interface for job management. The management website allows administrators to easily monitor the current state of the system and alter or add jobs. The heart of the system is the scheduler which runs periodic control loops used to update the system.

3.3.1 Communication Layer

The communication layer provides a SOAP interface for programmatically altering existing jobs or adding new ones. It also provides the interface through which clients can update their status with the server and receive work. The use of SOAP for communication with clients as well as other systems allows for third parties to more easily interface with the DOGMA system.

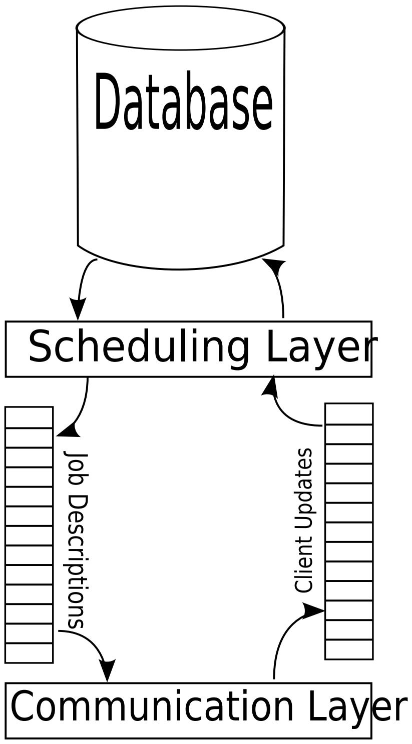

Figure 3.1: Communication Layer and Scheduler interaction through in memory data structures.

When clients connect to the server their information is stored in memory and later processed and persisted by the scheduler. Figure 3.1 shows the interaction between the scheduling and communication layers through two data structures which hold the information each stores for processing or us by the other. While communicating with clients the server looks at what the client currently reports it is working on and will provide different work, give more if the client has available processors, cancel what the client is currently working on, or have the client continue what it is doing. At no point during communication with the client does the server talk to a database or go to disk. This helps keep responses fast and allows the server to handle more clients.

3.3.2 Management Website

The Management Website provides an interface for humans to interact with the system. It provides information about currently running jobs as well as statistics about the state of the overall system including how many clients are currently connected and information about each. Administrators and job owners are also able to add jobs, edit or delete existing ones, or view the status if individual parts of a job.

3.3.3 Scheduler

The scheduling system's main purpose is to populate in memory data structures used by the communication layer when providing work to clients(Figure 3.1). These data structures are organized according to the OS and architecture of clients and contain all of the information needed by a client to start processing a job. The scheduling system is also responsible for processing client updates as well as removing inactive clients from the connected list and redistributing parts they were working on.

Chapter 4

Methods

Peer-to-peer applications are able to scale to large numbers of clients with little or no server resources required. By incorporating peer-to-peer techniques into DOGMA it will allow the system to scale well beyond its current abilities. Specifically by introducing peer-to-peer techniques into client server communications it will allow the clients to form clusters. These clusters will then use a single member, the aggregator, to communicate with the server instead of each doing it individually. While adding clustering abilities did not necessitate any changes to the DOGMA server, the opportunity was taken to simplify pieces of it as well as add optimizations. The DOGMA client was left largely the same with changes only being made in the communication layer.

4.1 Server

Though it was not required, a complete rewrite of the server was done. The rewrite was completed using updated technology which allowed for a reduction in the number of lines of code as well as the complexity of the system. The updated technology should also allow for greater scaling abilities, however no tests were done comparing the old and new system.

One of the simplifications to the system was the reduction of the number of client message types. There were three different types of messages one for client start up, one for updates, and finally one for client shutdown. These three were reduced to a simple update message. While reducing the number of message types, the server was also altered to allow more than one client to be updated in a single communication packet.

Each update message contains all of the information the server needs to know about the client, including the amount of memory, the type of processor, and the operating system currently running. The type of processor and operating system are used to determine what jobs may be run on the client. The mac address of the network interface used to communicate with the server is also included in the message to allow the server to uniquely identify the client.

Another important change to the server was the elimination of database actions required when a client communicates with the server. All client messages to the server are put into a queue which a control loop runs through every five seconds. Messages are removed from the queue and processed, updating information about individual clients as well as possible updates to the tasks and jobs they are running.

A different control loop acts to provide work for the clients. It does so by filling platform specific queues with information regarding available tasks. These queues are accessed when clients perform updates to provide additional work when needed.

4.2 Client

The DOGMA client changed very little with the only significant changes occurring in the communication layer. The DOGMA client is modularized allowing different pieces to be easily changed. The DOGMA Communicator is the client code which deals with communications with the server. To instrument clustering, only this code was replaced.

The new clustering communicator consists of two separate parts: the aggregation server and the client communicator. The aggregation server contains two data maps. One map contains the updates being sent from the clients to the server. The other map contains the server responses. A control loop communicates with the server a little more often than a regular client would. Right before communicating, the client update map has all of its data added to an array and is then cleared. This ensures that stale updates are not sent to the server and allows clients which have gone down to time out and have any of their assigned work redistributed to other clients.

The array of updates is sent to the server in one SOAP message. The returned message may contain an array of one or more responses, or nothing. The responses are separated out and put into the response map, whose entries are accessed and removed as clients perform updates with the aggregation server. If a client has a message put into the response map any messages it has put into the update map are removed so that the client can react to server commands and not have an inaccurate update sent to the server, this prevents the client from receiving duplicate server commands. When a client communicates with the aggregation server, if the update map contains an entry for that client it is replaced instead of later sending both updates.

The client communicator performs a couple of steps whenever the client wants to send an update to the server. It first checks to see if it already knows of an aggregation server. If it does, then it attempts to connect to it and upon success proceeds using that aggregation server. If the client is unable to connect to its known aggregation server or it does not have a known aggregation server it attempts to locate one. To take advantage of locality, the client only wants an aggregation server which is near by. Figure 4.1 illustrates the steps a client takes. First the client sends a UDP broadcast on its local subnet enquiring for any active aggregator. If it receives a response, the client proceeds using the newly found aggregator. If no aggregator can be found the client starts a new aggregation server in a separate thread and connects to it.

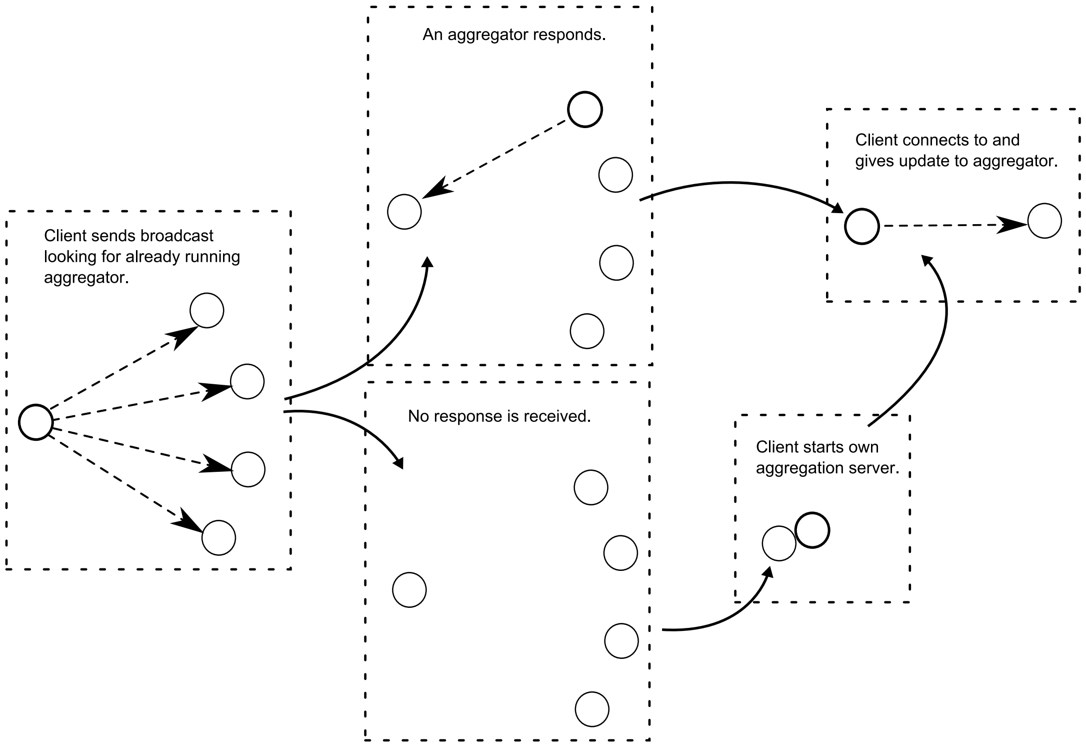
An aggregator responds.
Client connects to and gives update to aggregator.
Client sends broadcast looking for already running aggregator.
No response is received.
Client starts own aggregation server.

Figure 4.1: Client Prepares to Update

Once the client has an aggregation server to communicate with it begins following the update cycle shown in Figure 4.2. First clients send their updates to their aggregation server and receive back any pending messages from the server. Next the aggregation server performs an update with the server including all current messages from clients connecting to it. The server then sends back any responses which are put into the aggregator's response map and given to clients when they next connect to update.

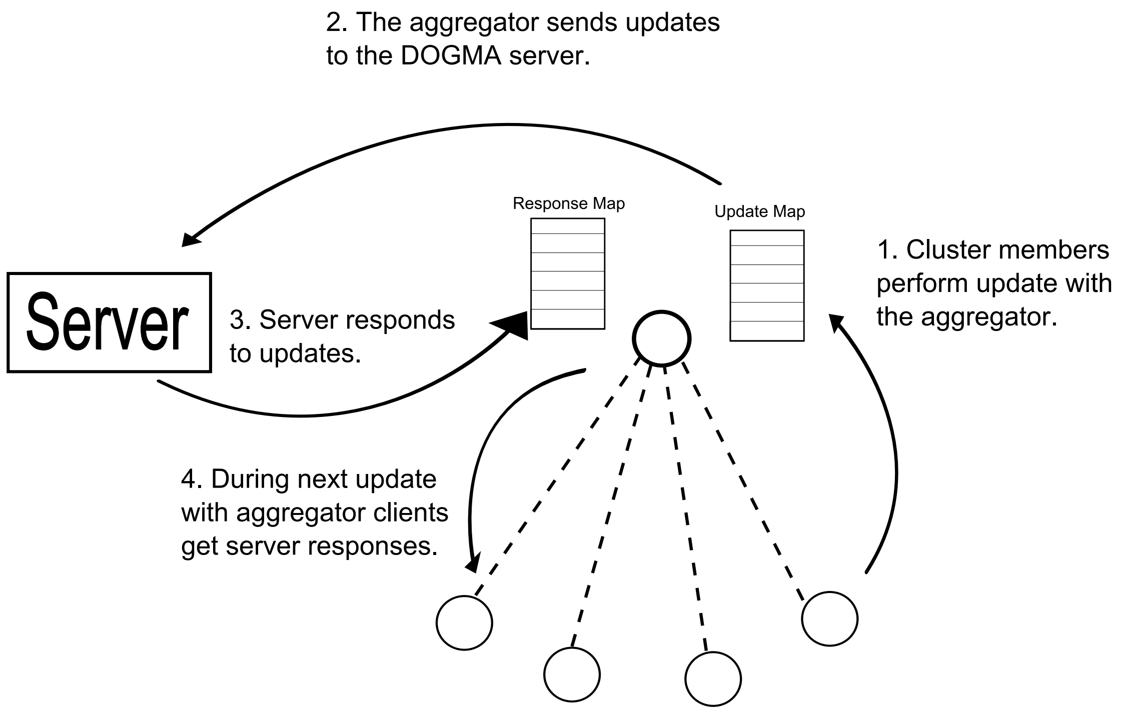
2. The aggregator sends updates to the DOGMA server.
Response Map
Update Map
1. Cluster members perform update with the aggregator.
Server
3. Server responds to updates.
4. During next update with aggregator clients get server responses.

Figure 4.2: Client Update

Chapter 5

Validation

5.1 Experimental Setup

To prove the usefulness of clustering, tests were run on clustering and non clustering clients. The tests consisted of starting up and running a group of clients for 10 minutes with and without work in the server queue. Tests with work had enough jobs and app-parts to keep all clients busy for the duration of the test. Because work was available job descriptions were passed to the which generated more network traffic. Tests were run starting with five clients and incrementing by five clients up to twenty-five clients. It took approximately 4 hours to run a complete set of tests.

Measurements were taken of total network usage at each client. At the server the network usage was monitored as well as the process CPU usage. It should be noted that when clustered clients were running, only one client acted as an aggregation server with all other clients connecting to it.

5.2 Results

The results exceeded expectations and clearly demonstrate the usefulness of adding clustering abilities to DOGMA clients. Below are shown the results for tests run with five and twenty-five clients. Appendix A contains a complete list of results for all tests.

5.2.1 Network Usage

Network usage was measured at the clients and the server. Usage was measured by querying the Linux kernel with measurements taken every second. We were not able to isolate the network usage for the individual process so network measurements were for the total traffic entering and leaving each machine. This makes it impossible to determine the exact usage of the client software but still provides enough information for conclusions to be drawn about clustering versus non-clustering client network usage.

Adding clustering significantly reduces the amount of network traffic at the server. Keeping the majority of network traffic generated by the system on local networks between clustered clients instead of crossing network boundaries talking to the server.

Figure 5.1 shows the network usage at the server with five clients connecting. The top graphs show usage when there are no jobs available to work on. This is useful information because it shows how much network resources will be used even when there is nothing useful happening on the system. The bottom graphs show the network usage when the system is working on something. You can see by looking at the top graphs that there is no significant difference in network usage when the system has no work to provide to clients. When there is work and the server needs to provide that information to clients, as well as the clients continually needing to tell the server what they are working on, there is a significant network savings when clients are clustering, as evidenced by the bottom graphs.

When there are more clients connecting the network savings become even more significant. Figure 5.2 shows the same information as figure 5.1 except with 25 clients connecting to the server. The benefits of clustering can easily be seen. Where before there was no significant difference in network usage when the system had no work, the top graphs show a significant savings in network traffic. When the

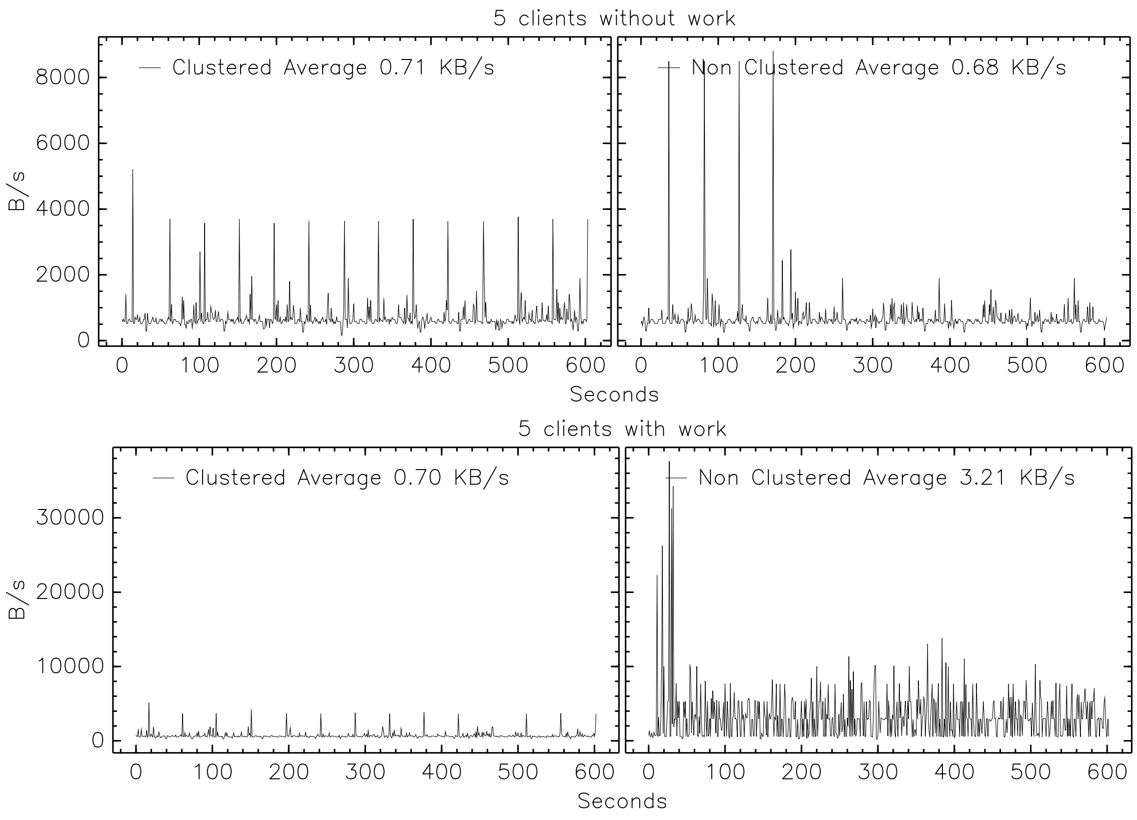

Figure 5.1: Server Network Usage: Reduced traffic when clustering and the system is doing something useful.

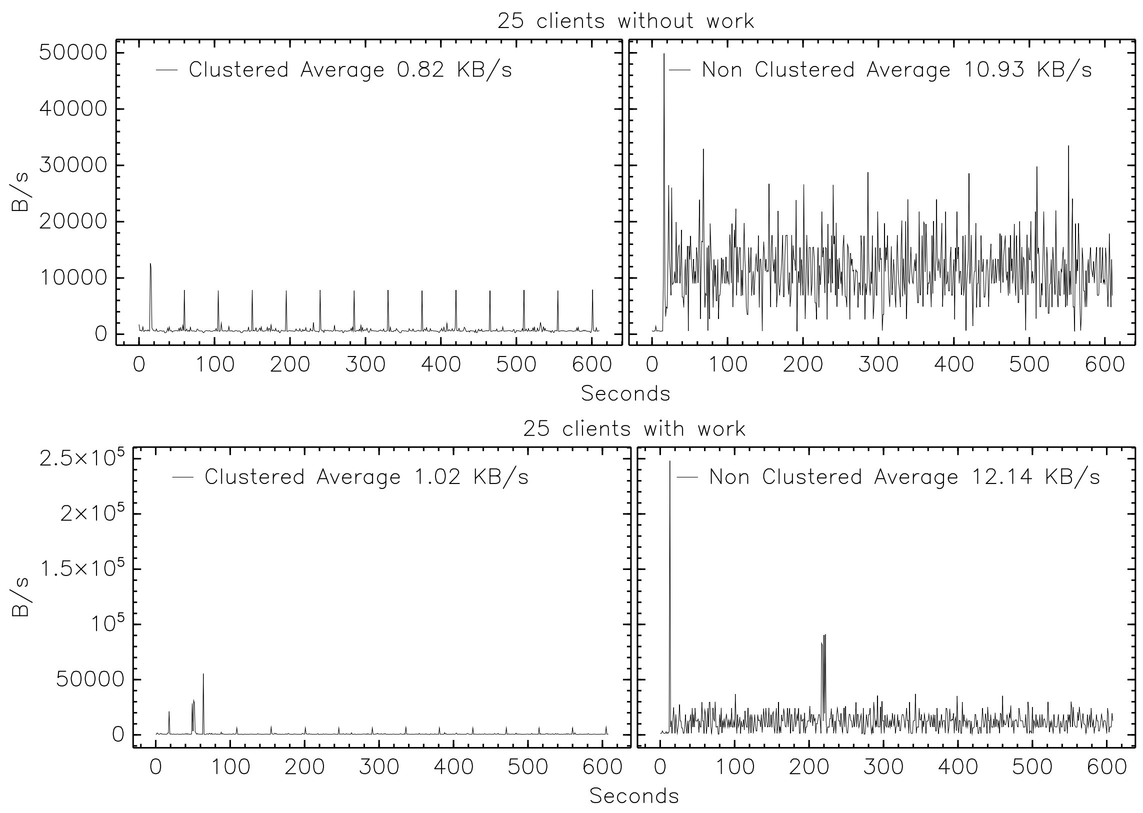

Figure 5.2: Server Network Usage: Significant reduction in network traffic when clustering with higher numbers of clients.

system is performing work there is even more of a savings. The bottom graphs show a difference of almost twelve times the network usage when not clustering.

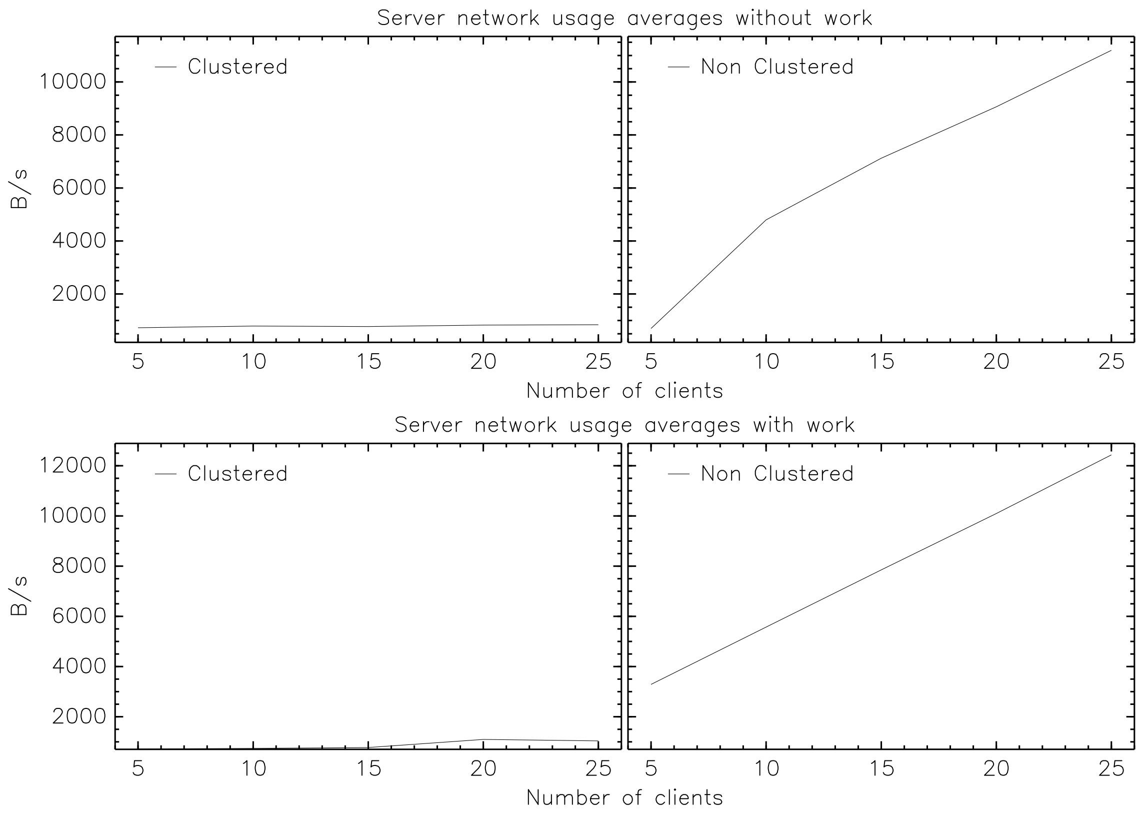

Figure 5.3: Server Network Usage Averages for 5 to 25 Clients: Non clustering clients have much steeper lines.

When we look at how average network usage at the server changes with increasing numbers of clients, as in figure 5.3, we can see how quickly network usage increases when clients are not clustering while when they are clustering there is a very gradual increase. Extrapolating the data for non clustering clients is fairly simple and if we assume a linear progression. Doing so for 1000 clients suggests an average network usage of more than 40 MB/s whether doing work or not. Doing the same linear extrapolation for clustering clients is more difficult because in our experiments we only ever had one aggregation server. We cannot assume that it would remain linear and would expect more of a stair stepping pattern as more aggregation servers are added. If we do assume a stair stepping pattern and use that for extrapolating to 1000 clients with cluster sizes of 25, average network usage would only be 3-4 MB/s.

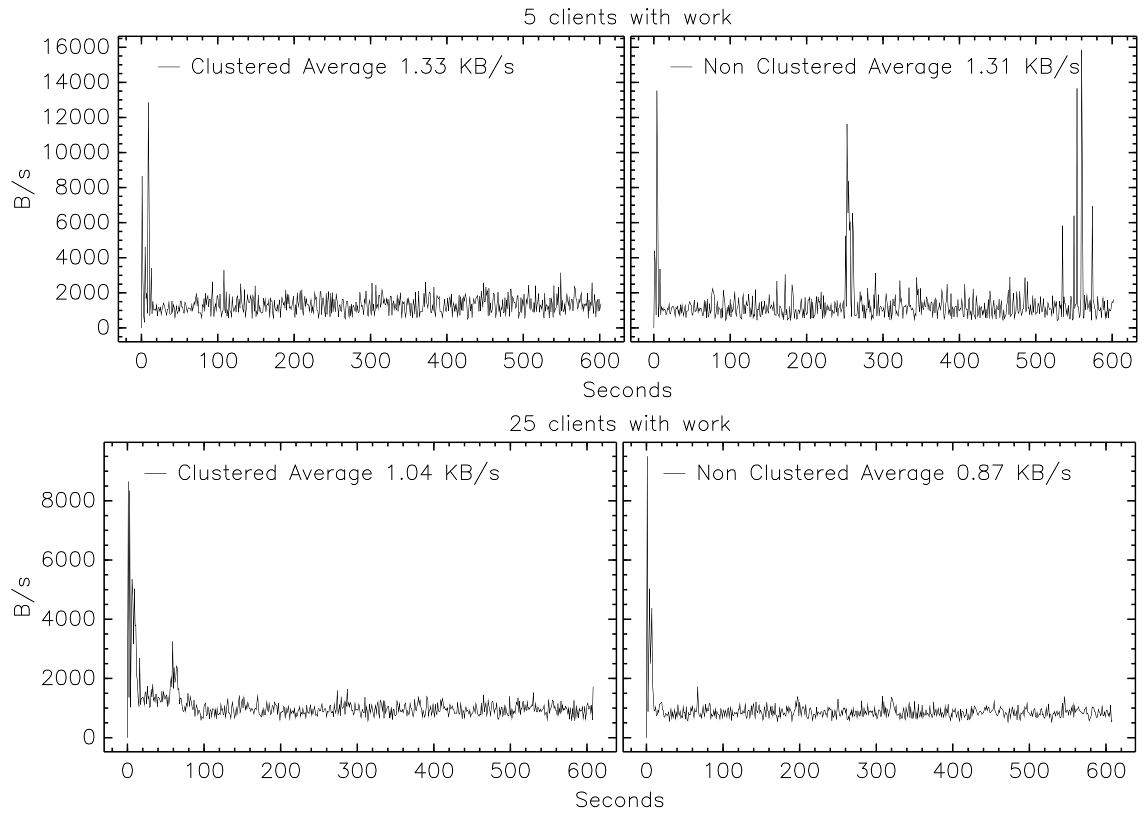

Figure 5.4: Average Client Network Usage: No significant change.

With such a significant reduction in network traffic at the server we expected to see a large jump in traffic at the clients, however that was not the case. Figure 5.4 shows the average network usage at the clients with five and twenty five clients. In both cases we are only showing the average traffic when the clients have work because that is when they would be sending the most information back and forth. In all cases however there was no significant change in network usage.

5.2.2 CPU Usage

Processor usage statistics were only taken at the server. Measurements were taken every second using the ps command in Linux. Instead of getting what percentage of the processor was used for the process, the total number of seconds was recorded. Figure 5.5 shows the total number of seconds consumed by the server process for five to twenty five clients. The top graphs show the total number of seconds consumed for clustered and non clustered clients receiving work. The bottom graphs show the same as the top except without work. The top and bottom graphs show the same thing. As the number of clients increases there is no significant change in CPU usage with clustering clients. Without clustering however the CPU usage increases almost linearly with the number of clients.

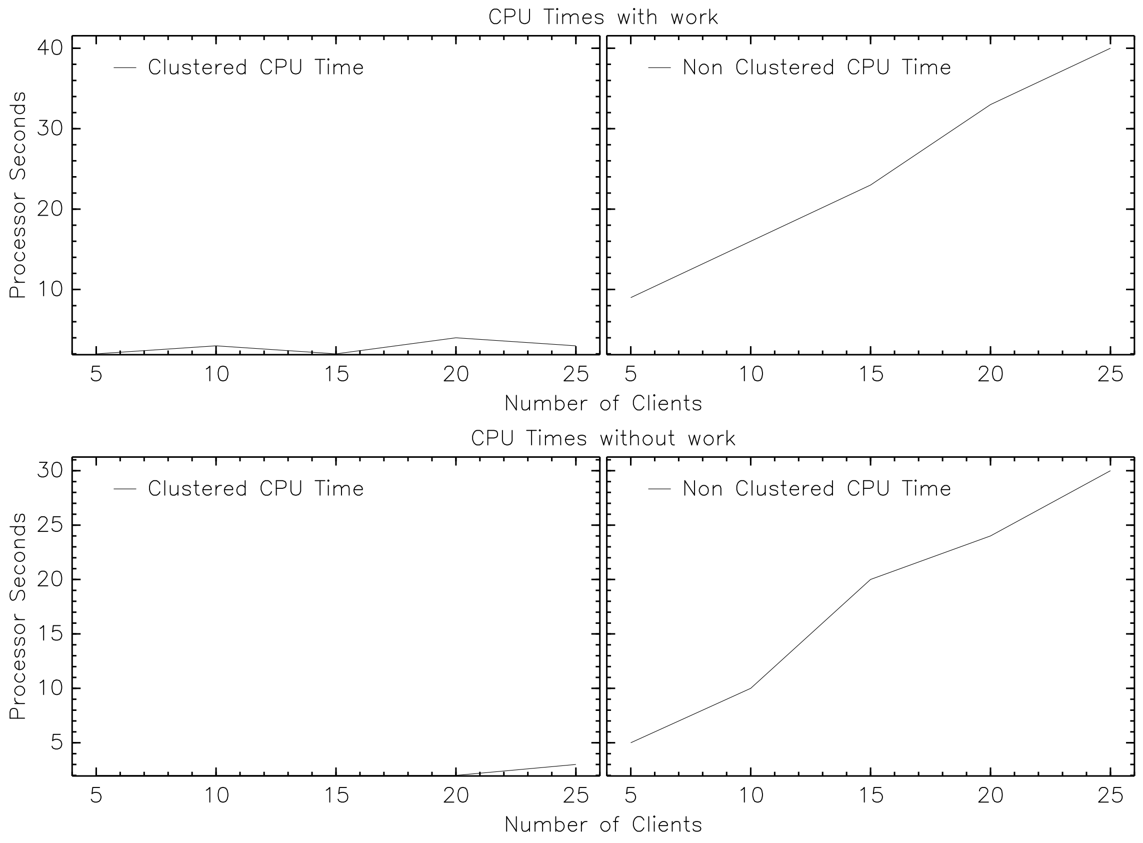

Figure 5.5: Server CPU Usage: Clustered CPU usage fairly constant. Non Clustered CPU usage increases almost linearly.

If we again extrapolate the data to 1000 clients, without clustering we would expect the server process to require at least 1200 seconds. That equals 20 minutes of processor time. Considering that each test ran for only 10 minutes we expect that the server process would require more time than the system could actually provide. Using extrapolation we would expect the server to be able to handle at most 500 clients.

Extrapolating the clustered data is again more difficult. If we assume a stair stepping pattern once more, 1000 clients in clusters of 25 would require 120 seconds of processor time or about one fifth of the total time available. This means that a single server running with clustering clients, instead of being able to handle only 500 clients would be able to handle closer to 5000 at once. This of course assumes that clients cluster into groups of 25. If the average cluster size were smaller we would expect the max number of clients to be lower as well, while with larger clusters sizes the reverse would be true.

5.3 Summary

Adding clustering abilities to the clients is beneficial to the scalability of the DOGMA system. Clustering reduces network traffic at the server with no significant changes at the client. Processor benefits from clustering are even more pronounced than network benefits. As the number of clients increases there is no significant change in the amount of CPU time required by the DOGMA server.

Chapter 6

Contributions and Future Work

6.1 Contributions

There are many distributed computing platforms available. However, none are able to both handle generic work and scale dynamically. DOGMA is able to handle generic work but fails to scale to the desired level because of the bottleneck created by constant communication with the server. By adding peer-to-peer abilities to the clients for communicating with the server they are able to form clusters and aggregate their communications through a single member. By aggregating communications with the server through single member the load on the server is greatly reduced and DOGMA will be able to scale to much larger numbers of clients.

Testing showed that even with only few clients there was a noticeable difference in the load generated on the server when clients were clustering and aggregating their communication. Both the network and processor usage were significantly reduced when clients clustered.

Network usage at the server rises much more slowly when clients cluster. With twenty five clients clustered network usage is one tenth non clustered usage. CPU usage follows a more extreme trend with total CPU time remaining fairly constant when clustering and increasing numbers of clients. While CPU time with non clustering clients increases linearly.

6.2 Future Work

6.2.1 Security

Security is important to consider in any system and DOGMA is no exception. Without proper security, the DOGMA system could be easily taken over and become another bot-net. Currently the only security is on the server and involves a simple login procedure. This was deemed adequate previously because it was assumed DOGMA was running on a secured network, meaning an attacker could not take over the network and pretend to be the DOGMA server. The introduction of clustering, however complicates things.

Clustering allows for a new kind of attack which, while not able to take over the entire system, allows for the hijacking of clients. An attacker with access to a DOGMA subnet could pose as a cluster aggregator. Once clients are checking in, the fake aggregator could hand out any sort of work it desired pretending it was from the DOGMA server.

Clustering introduced changes in communications sent between the client and server. Multiple messages are easily aggregated together. The server also now expects messages from multiple clients to come through at a single time and sends messages meant for multiple clients in response. Messages meant for different clients are self contained however and simply appended together. This allows for the introduction of a simple, yet robust security model for client server communication. A single field could be added to the client messages allowing the server to sign each one. Using a public-private key architecture clients would have a copy of the servers public key and the server could perform a hash of the message sent and then sign the hash. Clients could then easily verify that each message received did in fact come from the server.

6.2.2 Long running app-parts

Boinc and Folding@Home are fundamentally different systems from DOGMA. Instead of constantly monitoring clients they hand out the same work to multiple clients and then only expect to receive information back from those clients when the work has been completed. For long running jobs with app-parts that take a long time to process, this type of running environment works well.

With DOGMA's new message structure, it should be easy to allow some jobs and clients to run the same way. An obvious draw back to this method is that work is repeated by clients however, it also potentially increases the client pool to include ones which are only occasionally able to connect and report in.

6.2.3 Acquiring a non-local Aggregator

One weakness of the current clustering client is that it will only change its aggregator if the aggregator becomes unreachable. This causes clients that have started their own aggregator to never connect to another even if there are multiple available on the current subnet. Such a situation can occur if the network was overtaxed at the time the UDP aggregator search broadcast went out or if all of the clients were starting up at the exact same time.

It could be useful for clients who are connecting to a locally started aggregator to occasionally search for an aggregator on the subnet and if one exists, besides theirs, connect to it instead. By increasing the cluster size more messages are pushed through a single aggregator which reduces server usage.

6.2.4 Message Optimization

For simplicity, messages in an aggregation are completely self contained. This is fine for messages from the client to the server. Messages from the server to the client however may contain repetitive information. If the server has assigned multiple clients with the same job but different parts the information regarding what files to download will be repeated. It may be useful to change how aggregate messages are structured so that all repetitive information is sent only once and then let the aggregator split it out for the clients connecting to it.

Bibliography

[1] J. C. Ekstrom, "Local url resolution protocol," Master's thesis, Brigham Young University, July 2006. [Online]. Available: http://contentdm.lib.byu.edu/ETD/image/etd1436.pdf

[2] "Boinc," University of California Berkley. [Online]. Available: http://boinc.berkeley.edu/

[3] S. M. Larson, C. D. Snow, M. R. Shirts, and V. S. Pande, "Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology." *Computational Genomics*, 2002. [Online]. Available: http://fah-web.stanford.edu/papers/Horizon_Review.pdf

[4] "Xgrid, the simple solution for distributed computing," Apple Computer, Inc., Tech. Rep., 2005.

[5] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 2-4, pp. 323–356, 2005.

[6] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 3, pp. 200–222, 2001.

[7] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr., "Legion: The next logical step toward a nationwide virtual computer," University of Virginia, Tech. Rep. CS-94-21, 8, 1994.

[8] K. Fritsche, J. Power, and J. Waldron, "A java distributed computing library," 2001. [Online]. Available: http://www.cs.may.ie/~jpower/Research/Papers/2001/pdcat01.pdf

[9] T. Keane, R. Allen, T. Naughton, J. McInerney, and J. Waldron, "Distributed java platform with programmable mimd capabilities," 2003.

[10] A. Page, T. Keane, R. Allen, T. J. Naughton, and J. Waldron, "Multi-tiered distributed computing platform," in *PPPJ '03: Proceedings of the 2nd international conference on Principles and practice of programming in Java.* New York, NY, USA: Computer Science Press, Inc., 2003, pp. 191–194.

[11] "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004, pp. 137–150.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," 2002.

[13] G. Geist, J. Kohl, R. Manchel, and P. Papadopoulos, "New features of pvm 3.4 and beyond," *PVM Euro Users' Group Meeting*, no. September, pp. 1–2–10, 1995.

[14] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr., "A synopsis of the legion project, Tech. Rep. CS-94-20, 8, 1994.

[15] "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing.* Washington, DC, USA: IEEE Computer Society, 2000, p. 377.

[16] K. Aberer, P. Cudre-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid: a self-organizing structured p2p system," *SIGMOD Rec.*, vol. 32, no. 3, pp. 29–33, 2003.

[17] K. C. Tan, M. L. Wang, and W. Peng, "A p2p genetic algorithm environment for the internet," *Commun. ACM*, vol. 48, no. 4, pp. 113–116, 2005.

[18] "Grid mp platform, 4.2 architecture overview." [Online]. Available: http://www.ud.com/resources/files/mp_architecture.pdf

[19] N. H. Ekstrom and J. J. Ekstrom, "Dogma: An open source tool for utilization of idle cycles on lab computers," in *Proceedings of the 2005 American Society for Engineering Education Annual Conference & Exposition.* American Society for Engineering Education, 2005.

Appendix A

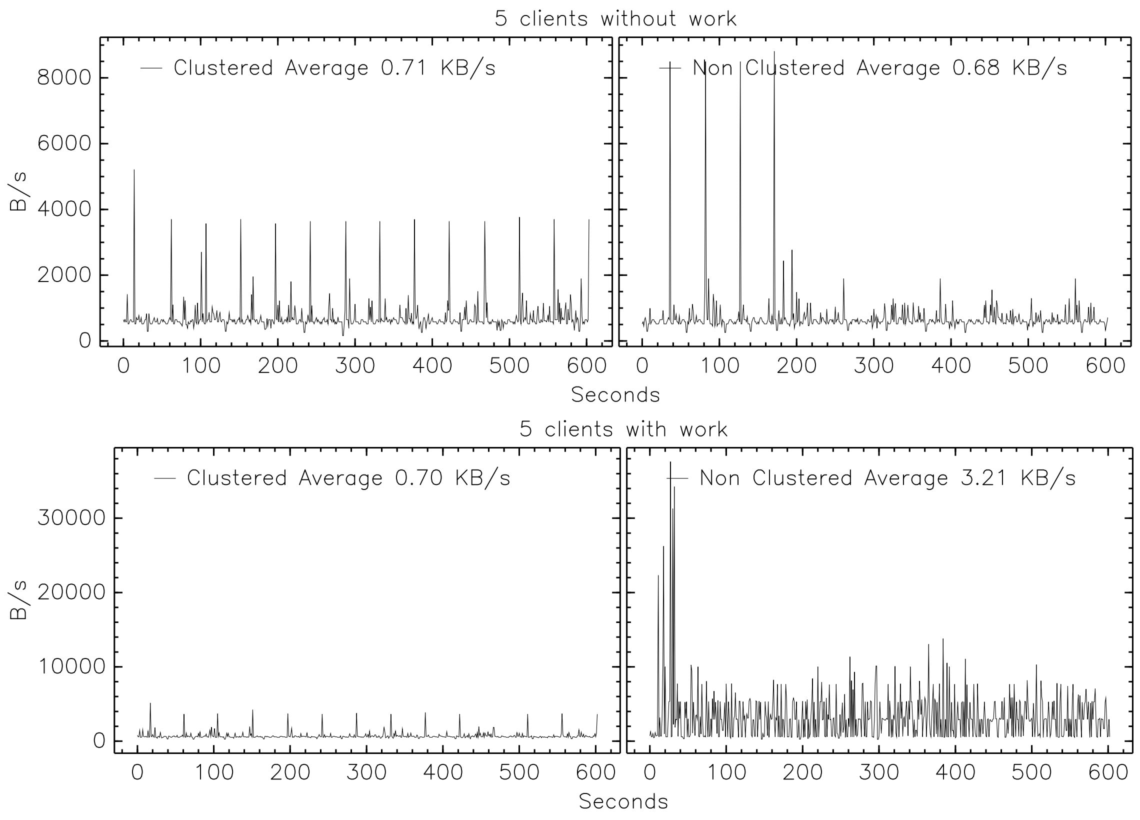

Figure 6.1: Server Network Usage: 5 Clients.

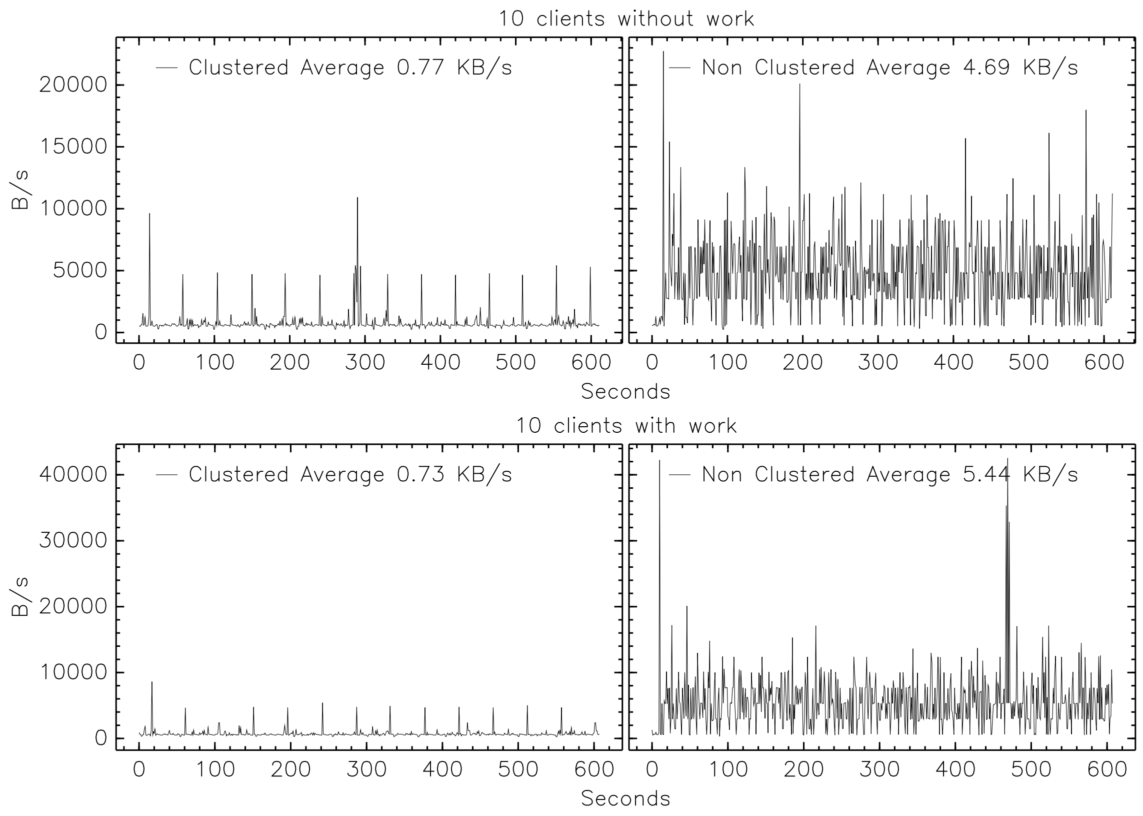

Figure 6.2: Server Network Usage: 10 Clients.

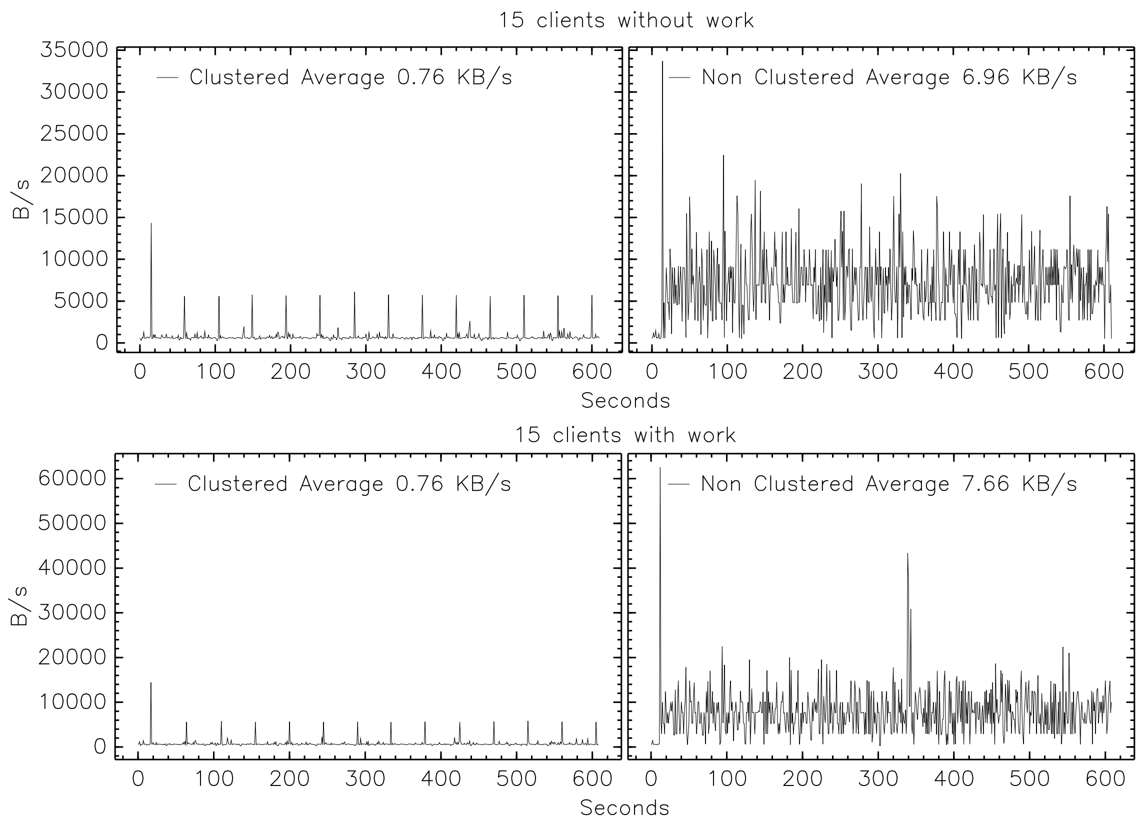

Figure 6.3: Server Network Usage: 15 Clients.

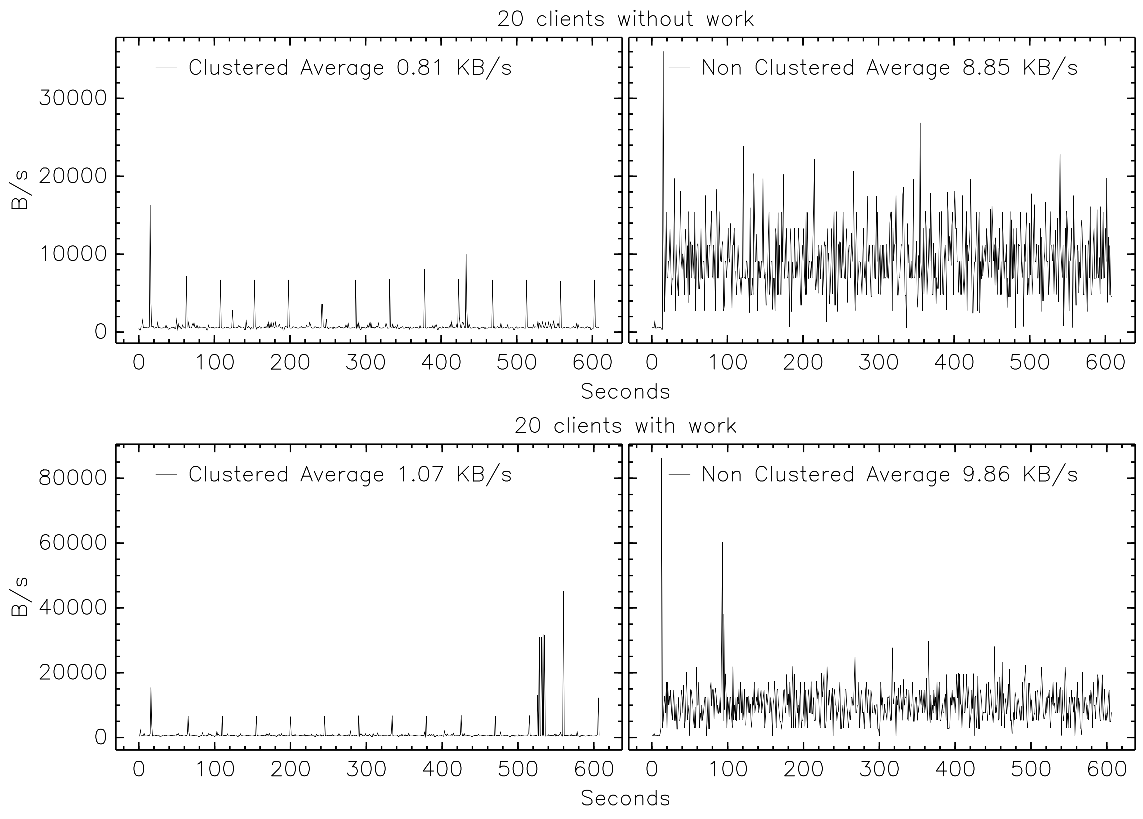

Figure 6.4: Server Network Usage: 20 Clients.

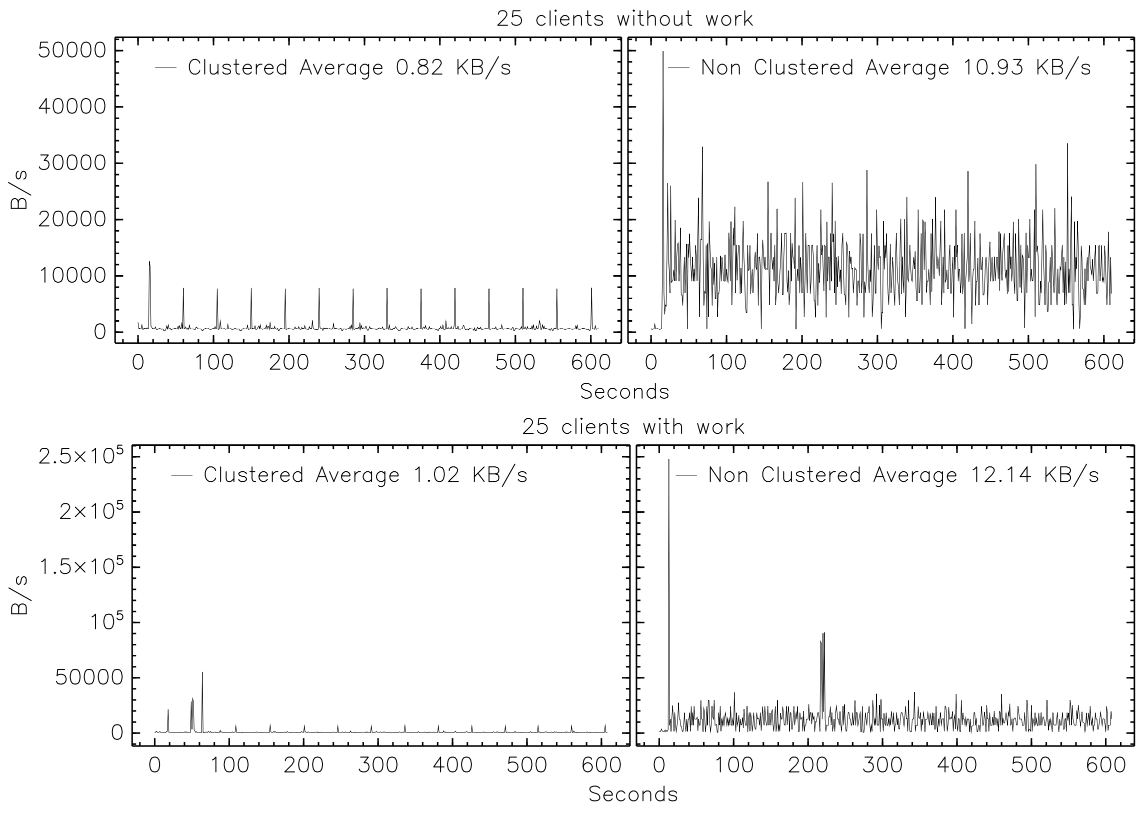

Figure 6.5: Server Network Usage: 25 Clients.

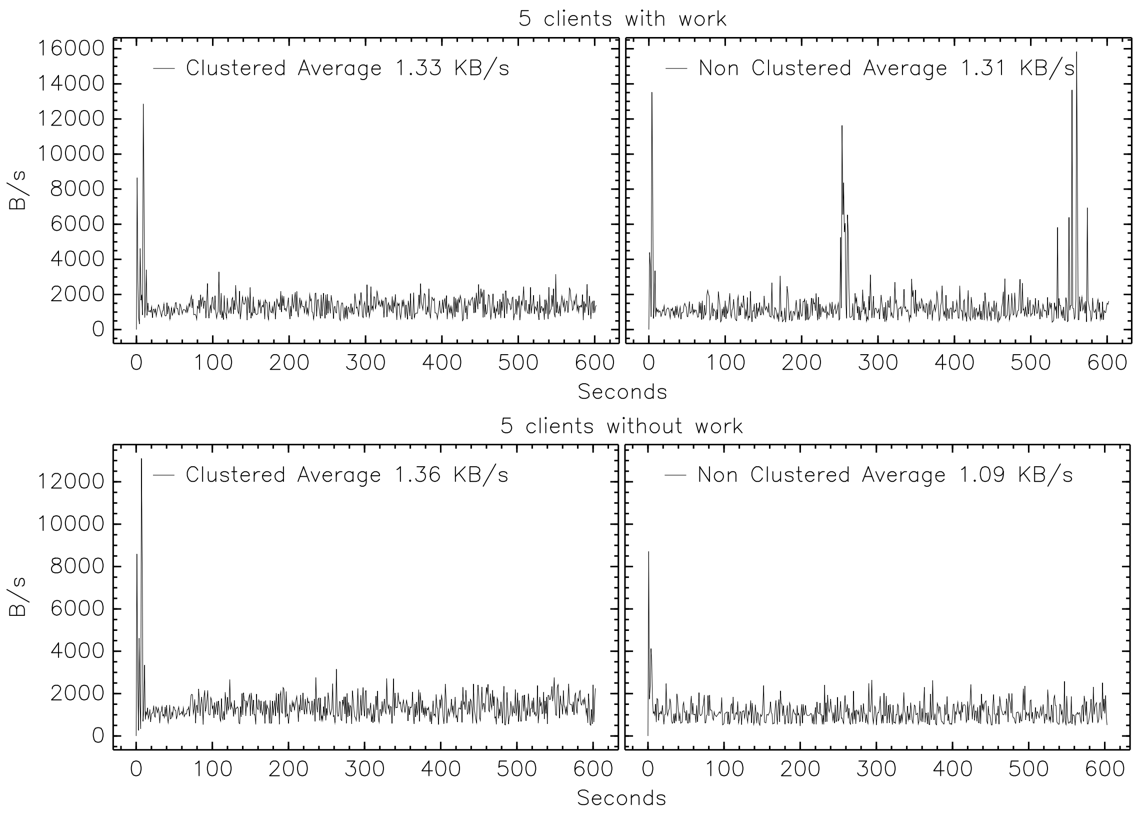

Figure 6.6: Average Client Network Usage: 5 Clients.

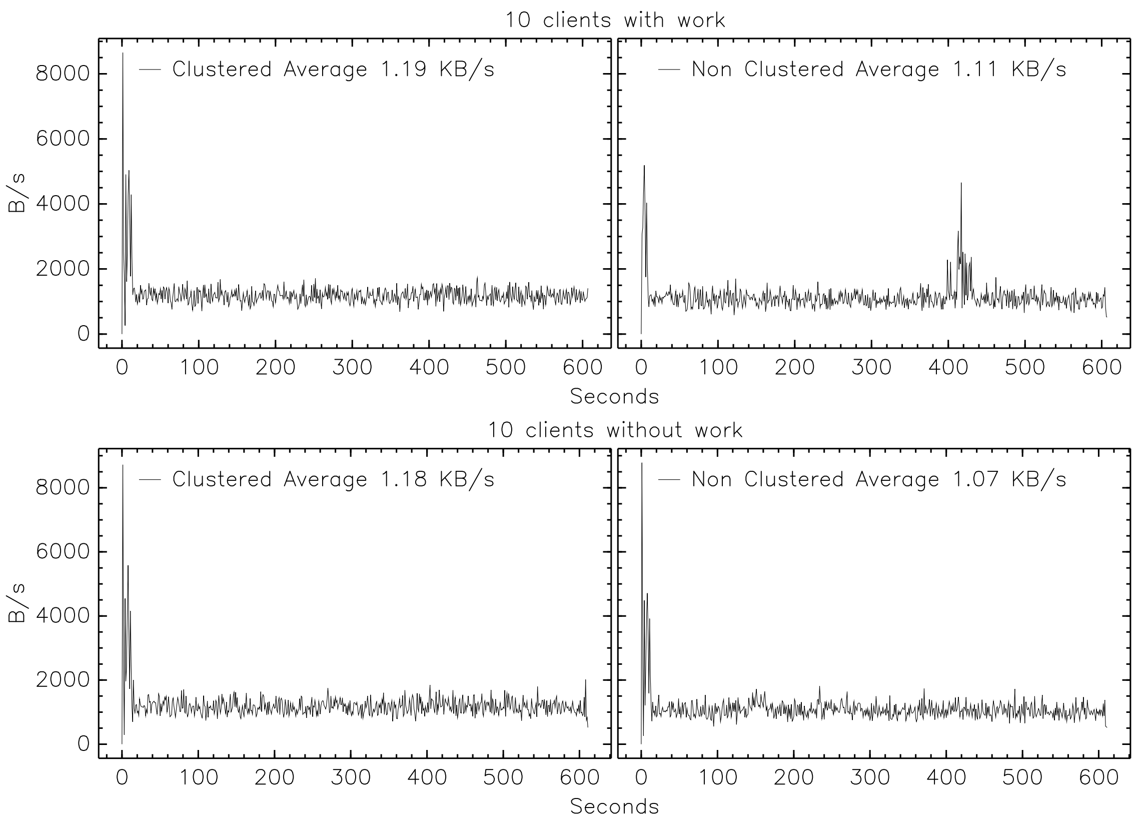

Figure 6.7: Average Client Network Usage: 10 Clients.

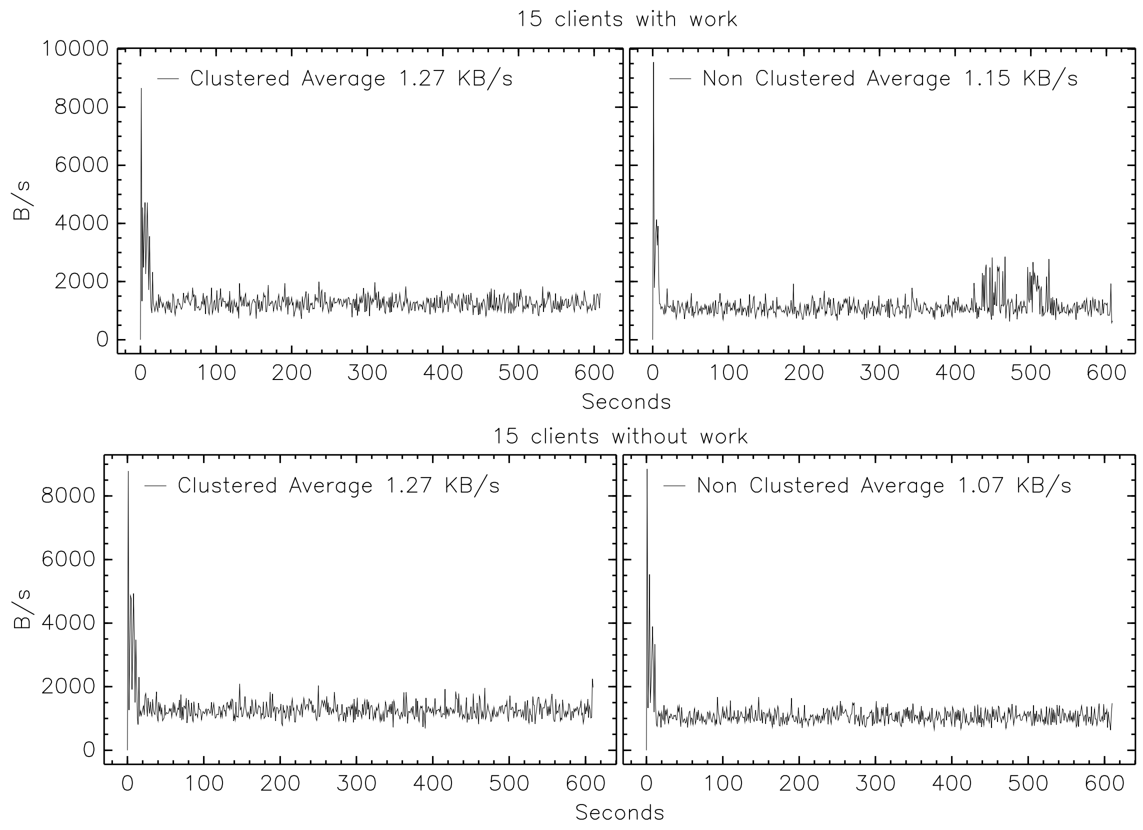

Figure 6.8: Average Client Network Usage: 15 Clients.

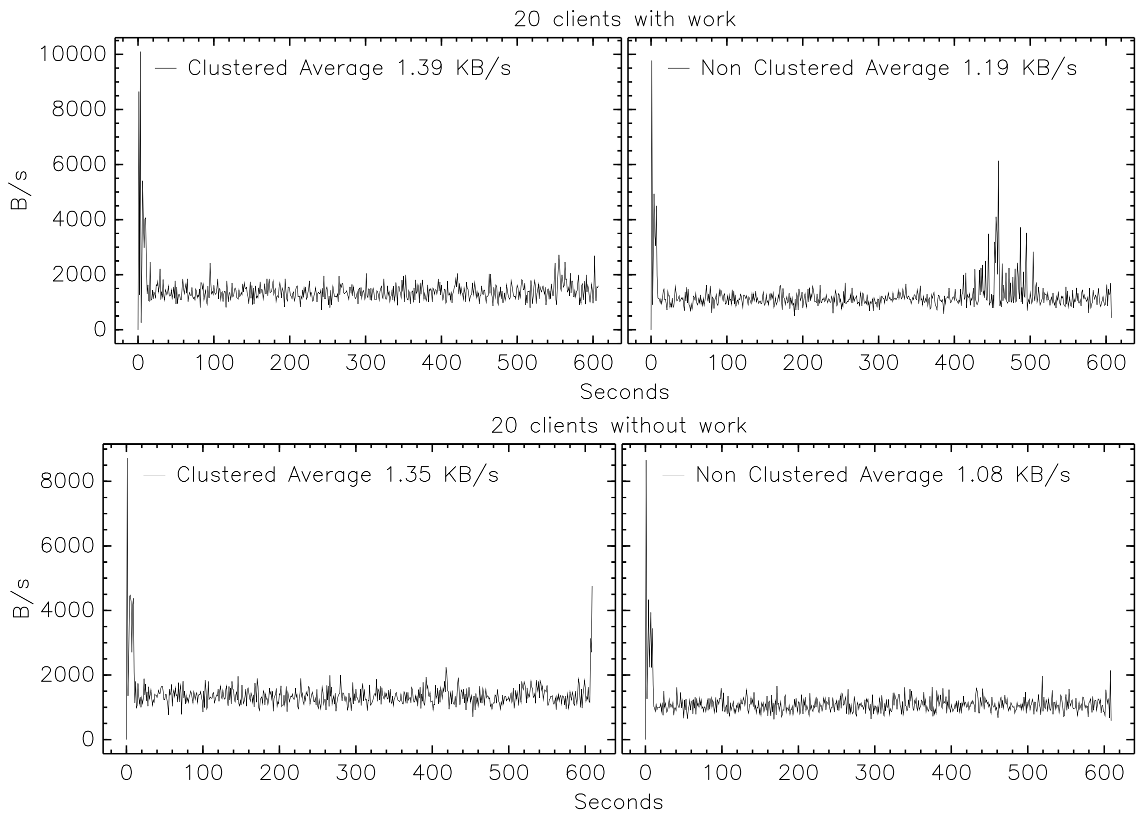

Figure 6.9: Average Client Network Usage: 20 Clients.

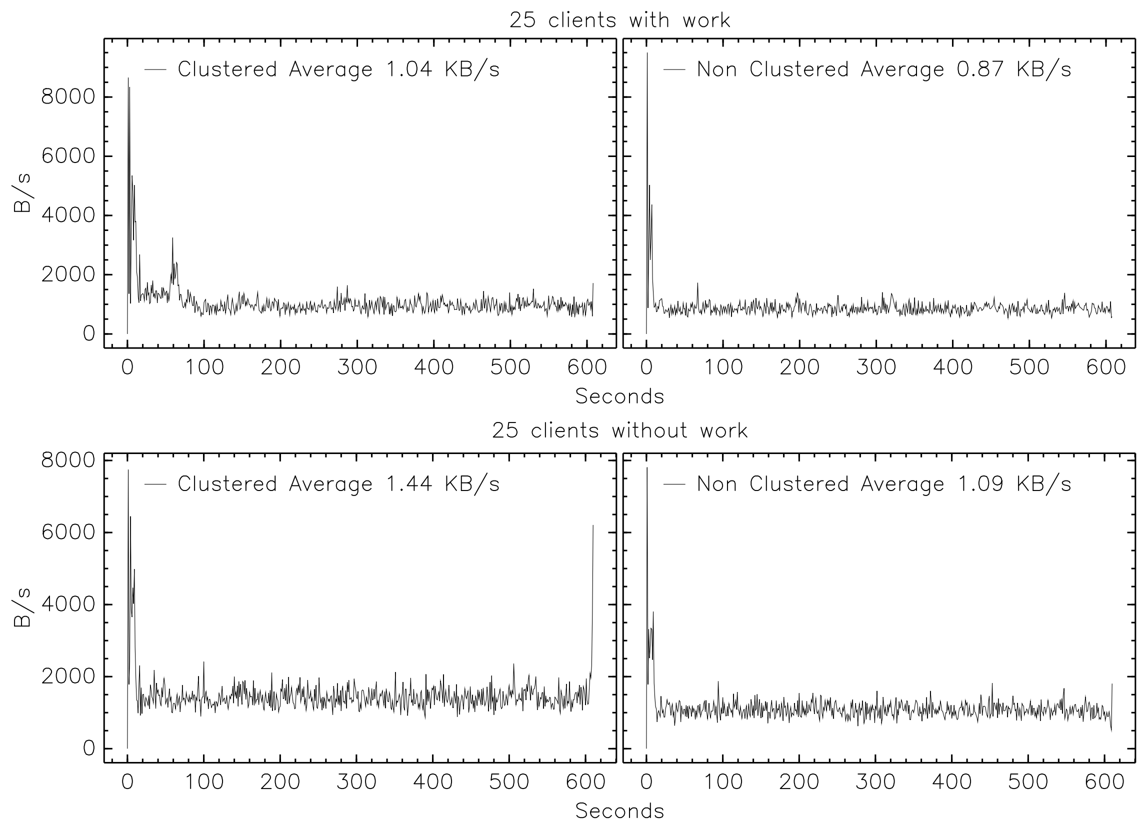

Figure 6.10: Average Client Network Usage: 25 Clients.