THE AMERICAN UNIVERSITY IN CAIRO
SCHOOL OF SCIENCES AND ENGINEERING

Multi-Channel Clustered Web Application Servers

A Masters Thesis
Department of Computer Science and Engineering

Status Report
Seminar II

Submitted by Karim Mohamed Sobh
B.S., Computer Science
The American University in Cairo, 1997

Supervised by Dr. Ahmed Sameh

December/2008
ABSTRACT

MULTI-CHANNEL CLUSTERED WEB APPLICATION SERVERS

As web application environments, standards, protocols, and development technologies become more mature, such environments become more dominant and more appealing to software application engineers and creators. Web application environments do provide open standards and well published interfaces that allow easy deployment, expandability, integrability, and mission critical servicing. As much work has been done on the standardization of interfaces, and inventing new development models to serve such standardization as well as other important features like expandability, extendibility, high availability, and business logic isolation, performance has always come later in priority. Performance was always left to be handled by the advances made in the field of hardware, either in processing components like semiconductors, or in communication hardware protocols. Few approaches have been taken in the direction of performance from software perspective. Most web application environments have been built on top of the current backbone architecture of web environments. In this thesis, web environments are analyzed in detail, by looking at low level components of the environment architecture, and questioning the benefits of basic building blocks and whether they are designed and implemented to provide optimal performance. We believe that the current building blocks of web environments are used as standardized components to build on top of, believing that they provide the best performance, ending up with a very nice creature, which has a lot of virtues but lacking performance; better performance can be obtained with some changes to its building blocks. As web environments are very widely used by millions of users and client applications, it is our target to make such architectural changes in the most delicate transparent way so that already existing applications will not suffer compatibility problems, yet achieving a gain in performance. In addition, standards, concepts, and architectures will be borrowed to construct these changes in different ways aiming at improved performance, and features such as high availability and fault tolerance, in addition to attempting to eliminate the difficulty of usage and deployment of web applications.
# Table of Contents

1. Overview................................................................................................................................................1
2. Motivation and Problem Definition........................................................................................................3
   2.1. Motivation......................................................................................................................................3
   2.2. Problem Definition..........................................................................................................................5
3. Proposed Solution......................................................................................................................................7
   3.1. Proposed Architecture.....................................................................................................................7
   3.2. Research Question..........................................................................................................................10
   3.3. Expected Deliverables...................................................................................................................10
4. Work Done Before Thesis Proposal.......................................................................................................12
   4.1. Design...........................................................................................................................................12
   4.2. Prototyping and Implementation....................................................................................................12
   4.3. Experiments and Results................................................................................................................12
      4.3.1. Single Channel Exhaustive Experiment.................................................................................13
      4.3.2. Multichannel Delay Parallelization Experiments.................................................................15
      4.3.3. Observations and Result Analysis........................................................................................26
5. Work Done So Far...................................................................................................................................27
   5.1. Design...........................................................................................................................................27
   5.2. Prototyping and Implementation....................................................................................................27
6. Remaining Tasks and Time line............................................................................................................28
   6.1. Remaining Tasks...........................................................................................................................28
   6.2. Time line ......................................................................................................................................28
List of Figures

Figure 1: Proposed Multi-Channel Web Environment based on UDP .............................................. 9
Figure 2: Single Channel Results ........................................................................................................ 15
Figure 3: Multichannel With 0 seconds delay .............................................................................. 17
Figure 4: Multichannel With 4 seconds delay .............................................................................. 19
Figure 5: Multichannel With 8 seconds delay ............................................................................ 20
Figure 6: Multichannel With 12 seconds delay ......................................................................... 22
Figure 7: Multichannel With 16 seconds delay ......................................................................... 23
Figure 8: Bandwidth gain relative to increase in processing delay .............................................. 25
1. Overview

Computing power, defined as the amount of work that can be done in a given amount of time, which is meant by the expression “response time”, is one of the main features that is targeted in the computer science field. As software complexity increases and the amount of data that needs to be handled, demands for more processing power will continue. In this thesis the expression “Computing Power” will be replaced by computing performance. In fact, better computing performance is always a target, and we don’t mean by “Computing Performance” just processing performance, rather we mean the total computing throughput of a target system including processing, communication, buffering, caching, pipelining, ...etc. In more general terms we look at performance from the delivery of service aspect. Performance enhancement has always been sought through two approaches: hardware and software, which were always overlapping with a gray area that included both approaches.

Taking the hardware approach, better processors are invented, better semiconductor technologies are used, better scheduling mechanisms are utilized, better pipelining techniques are constructed, better hardware communication layers are deployed, and more. As the hardware technology approached the physical limits imposed by matter, e.g. the speed of light, incremental gains in performance on this front slowed. New techniques, like clustering, have emerged to make use of the already existing technology and its ongoing advances to get the needed performance. Clustering is a very wide area of research that computer scientists think is the solution to the continuing demand for better performance. Clustering can be implemented starting from a group of processors to its ideal maximized setup on a group of nodes, or physically isolated computer machines CPUs. In this thesis, clustering is one of the main techniques that is used to obtain performance in new advanced web architectures.

On the other hand, the software approach searches for better algorithms, invents code optimization techniques, designs approximation algorithms to solve scalable problems, utilized parallelism techniques inherited from clustered hardware, and invents new software development models to ease and extend the development of vast software application systems; for example the parallel and clustering programing models that enable utilizing clustered environment to virtually act as one entity through what is called Single System Image (SSI). Yet performance has always lagged behind the functional requirements, and performance tuning always came in later software releases as enhancements. As clustering technologies emerged, and the infrastructure on the hardware level was ready and mature, there was always a need to develop tools and libraries to be able to utilize the clustered environments to run algorithms and get better performance. This is a gray area where hardware and software approaches overlap. It is the responsibility of the cluster management software system, a middleware, to make the best use of the underlying clustered environment, and get as much performance as possible out of it. Basically, better middleware utilizes the underlying environment better and makes more use of its computing and storage resources. The expression “middleware” here encapsulates all the environment
components such as communication protocols, scheduling mechanisms, dispatching engines, resource handling, garbage collection, ...etc.

The web environment domain is becoming more popular and widely used to the extent that some visionaries predict that all the applications in the future will be implemented on the web or at least will have a web interface. Web environments have utilized clustering techniques to serve the massive transactions and extend performance to server mission critical needs and provide the best performance obtainable.

The aim of this thesis is to introduce changes in web architectures, investigate their impact on all ranges of web environments, and how they will affect performance. We claim that the current web architectures do not make full use of the available power of clustering techniques and hardware system resources due to limitations in the underlying web communication protocol. It is possible to change this protocol transparently in a way that will improve performance, and help techniques like parallelism, business logic isolation, dispatching, availability, and fault tolerance to better utilize the underused web environment resources of web clusters.
2. Motivation and Problem Definition

2.1. Motivation

As demand for web application servers has increased to handle exceptional performance tasks, and as web applications are becoming more complex, new techniques for true performance gain and full resource utilization are becoming crucial. As hardware approaches reach their limits due to physical boundaries, clustering and parallelization have become one of the most important means of achieving performance when building web application servers and web components through the horizontal expansion of web application deployment. Much effort has been expended over the past few years to invent new web development models as well as to develop new architectural and infrastructural concepts utilizing clustering techniques and making the best out of them to gain performance and features like extendability, expandability, re-usability, high availability, and fault tolerance. All the efforts in this field so far have founded their work on a backbone communication infrastructure based on connection oriented TCP.

The motivation behind this thesis is how to broaden our vision and change the communication protocol upon which our proposed web environment operates whenever beneficial, while keeping the standard client interface transparently unchanged. Changing the TCP communication protocol will allow us to provide a better infrastructure for high availability, fault tolerance, re-usability, using hybrid web components, and provide better caching schemes that cache static content as well as the partial static content provided by dynamic components.

This thesis is concerned less with inventing new technologies and concepts than with restructuring existing web components, architectures, and concepts, with as little changes possible to enhance performance by providing a better infrastructure for them to operate upon. Therefore, the proposed new architecture will borrow ideas such as functional decomposition and data level parallelism from MPI; explicit concurrency directives from OpenMP, divide-and-conquer master-slave and pipeline processing from parallel programming paradigms; caching and buffering from Parallel I/O; fault tolerance from PVM; extended web scripting techniques to provide in-line scripting parallelization; extended web service mechanisms to provide service state migration between different runtime environments; dynamic load balancing through calibrators from cluster and grid computing; and managed heterogeneity from cluster and grid computing.

It is important to emphasize that this thesis is not targeting the development of a new faster communication protocol for data streaming, rather it is targeting a better communication protocol to handle parallel execution of web components and parallelize processing delays in decomposable applications. It is also important to stress that the new architecture will provide its benefits in environments serving applications with considerable balanced amounts of both processing and I/O of decomposable applications by nature, rather than pure streaming I/O environments which belong to a domain of applications.
other than those tackled here.

The motivation for conducting this thesis has increased after conducting some preliminary proof of concept experiments to probe the validity of the ideas discussed. The results of the experiments were very promising. The experiments also shed some light on different architectural as well as implementation options that could be investigated in this research which made the ideas more mature.
2.2. Problem Definition

Throughout the background study conducted in this thesis and presented in the thesis detailed proposal, many variations in current web environments with their different ranges and setups are available. The common denominator, which should exist in any web environment on the network as well as the application layer, is the front-end communication protocol over which web environments share information. Although this front-end web interface, which is HTTP over TCP, provides standardization and compatibility between different entities communicating in a web environment, nevertheless some limitations related to its performance and error recovery remain.

Due to the persistent nature of TCP network communication, the smallest unit of execution in a web environment is the web transaction. A web transaction cannot be broken down to execute in multiple execution threads on both processing and communication fronts, from server to client end-to-end. There are many techniques nowadays for distributing the execution of a transaction service such as in EJBs (Enterprise Java Beans) environments, and for making different portions of the service run in parallel over multiple threads, which might be running on the same machine or over different nodes in a clustered environment. However, a problem arises from the fact that there is still a traffic gateway which collects the data returned from the server to the client and consolidates it before sending it to the client over the network. Since a web application service component is a running program, and since programs do some processing to generate some results, then the output of a web application service is based on execution in the first place, and hence if a service execution resulting from a web transaction initiation can be broken down, the delays in the execution of the service can be parallelized, which will provide better performance with respect to service response time and delivery. It is important to stress that the target environment that will benefit from fixing such a limitation is not one that is based solely on data transfer. Instead, it should be an environment that runs those applications that follow the pattern of interlaced processing and I/O.

For the same reason, which is the TCP persistence, a failing web transaction will require the re-initiation of the request which in most cases the web client will be aware of. A TCP connection is fundamentally a persistent connection between two partners that requires that the source and the destination be defined upfront before starting real data transfer between them, which means that if an HTTP connection failed, the client must re-execute this transaction by initiating a new TCP connection to the server. In addition, the HTTP protocol does not have any fault tolerance construct. It is left totally to the client application, which is in our case a web browser or a web client program, to report an error and take the appropriate action.

A major decision in web environments is request dispatching, which is either inefficient Layer-7 application level dispatching enabling integration with application logic, or a fast reliable Layer-4 dispatching which is difficult to integrate with application logic. Layer-7 dispatching is in some cases slow as it needs more than one TCP connection to function,
and in other cases inefficient as once it dispatches the first request coming from a client, the client is kept with the server it is routed to, in which case the dispatcher cannot control the load as a client routed to one server may have fewer subsequent requests than another client. A problem also arises from that fact that Layer-4 and Layer-7 benefits cannot be grouped in one dispatching scheme as the benefits of each are totally based on the basic concepts that each dispatching scheme is built on top, and they are fundamentally different.

Caching in web environments, either on the client side or on the server side, segregates content into two types; static content which is usually cached like static HTML files and images, and dynamic content which is not cached at all due to its being generated by a back-end engine. But actually some parts of the dynamic content that is generated by the back-end service each time the service is called is static, and the ability to cache such content would enhance the performance.

To wrap up, the weak points that are being investigated and tackled are:

1. Minimum unit of distribution and parallelism, end-to-end, is a web transaction.
2. A failing web transaction can be recovered only by re-executing the whole transaction, and the HTTP protocol does not support fault tolerance constructs.
3. Caching of content is according to the content type, either static or dynamic. In the case of dynamic content part of the content may be static which still cannot be cached.
3. Proposed Solution

The proposed solution to the three points defined in the previous chapter is to change the TCP communication layer and enable the web components discussed in the background chapter, of the thesis proposal, to communicate on another network protocol that provides the ability to execute services' portions in parallel, in the case of decomposable services. Also the HTTP protocol should be amended with extra constructs to ease the distribution, and utilize clustered alternatives of different portions of a service to execute distributedly and in parallel.

The difficulty with the proposed web solution, is how to provide communication as transparently as possible for the web client, by still providing the HTTP over TCP interface to the web client, and to overcome the problems mentioned in the previous chapter. As how a web environment is designed and implemented is not important as long as it conforms with the web interfaces and APIs, this can be achieved through proxying mechanisms between the client and the server. The most important decision is how and where to apply such techniques to get the desired result.

In the new proposed web environment, the TCP communication protocol will be replaced with the UDP communication protocol which allows changing the source of the HTTP reply data stream over the same socket, and serving a single transaction over a number of virtual UDP channels. To be able to do that transparently a new component will be added which will act as an agent which talks UDP to the web environment over multiple communication channels and will talk TCP to the web client over a single communication channel. This agent will be located on the client hardware and and will act as a proxy server between the web client and the UDP based web application server. As such, all the TCP communication will take place over the loopback address which is a network communication medium implemented over memory. Concurrency directives will be used in the new environment server-side scripts to identify different portions of the service that can run in parallel, and return their data stream replies over different UDP communication channels to the proxy agent client. The HTTP protocol will also be extended to give the proxy agent client a subset perspective of the cluster resources with respect to a specific service to be able to recover connections transparently from the client and without the re-execution of failing web transactions.

3.1. Proposed Architecture

The architecture of the proposed web application server is built up of two main components which are the Container, and the High Performance Agent (HPA). The container is a normal application deployment server which can load application component instances in the form of services, as well as provide the resources required for them to execute and function. The Container will support a UDP based communication layer through which all communication between any Container and its clients will be over a stateful communication protocol built on top of UDP. Now, an important question pops up which is
Why build a stateful communication protocol over UDP while the TCP protocol exists? The answer can be summarized in three points: 1) The TCP protocol is too general with a lot of overhead to accommodate its general features designed to serve any kind of communication sequences between any two entities. A minimized version of TCP can be implemented to remove all the overhead and be specifically for web transactions on a Request/Reply basis only. 2) As the tests presented in Chapter 5 show, over UDP more web transactions can be handled than over the normal TCP used in current web servers; thus utilizing concurrent channels to serve one web transaction from different container nodes. 3) A deviation from the TCP protocol is needed to be able to change the source of the data stream at any point of time. A container which is sending a web transaction reply to a specific client must be able at any point of time to delegate the execution of such web transaction to another container located physically on another container node which will resume the sending of the data stream, and hence the whole web transaction. This capability provides an infrastructure for fault tolerance through service takeover.

Since the Container will not be able to communicate except through a proprietary protocol based on UDP, and since normal web clients communicate with web servers using HTTP over TCP, an intermediate translator will be necessary to narrow the gap and enable the web client to transparently send its requests to the container. Thus, the High Performance Agent component is introduced which will be referred to throughout this document as HPA. Acting as a reverse proxy, the HPA will be located physically on the machine which the web client initiates its web requests from. Unlike any normal proxy, the HPA provides proxy operations between a web client and a Container over different communication protocols, so the HPA will be communicating with the web client through normal HTTP over TCP and will translate those client requests to the container through an extended HTTP protocol over UDP. The HPA is designed to be a reverse proxy because unlike normal proxies, a reverse proxy serves a specific destination or a number of destinations. In a realistic situation, the HPA is not considered an overhead, as it is located on the client machine, very tightly coupled with the web client and serves only the normal load of a single user's web transactions.

The following abstract diagram shows the proposed new architecture.
Figure 1: Proposed Multi-Channel Web Environment based on UDP
3.2. Research Question

The research question raised by this thesis is “what is the effect on performance of changing the communication layer of web environments from the normal stateful TCP to the stateless UDP communication protocol, and how will this change help the emergence of new mechanisms and features that will add to already established characteristics of web environments such as expandability, extendibility, fault tolerance, high availability, ... etc. ?”

3.3. Expected Deliverables

The expected deliverables of the new environment can be divided into two parts; first performance deliverables and second new functional features that the new environment will provide.

The expected performance deliverables are to get an up to 50% increase in the speed of web transaction execution end-to-end compared to Apache, and a 20% increase in the number of transactions that can be handled by the container in a given time frame compared with Apache. More importantly, the aim is an environment in which the performance gain increases proportionally with the increase of the amount of processing per service. It is very important to stress on the fact that those types of applications that are targeted are: 1) decomposable by nature, and 2) demand a considerable amount of processing compared to I/O.

Regarding the functional features deliverables, an environment implementation of the main features will be provided by the end of the thesis that supports the development languages, C++ and JAVA, to illustrate the following functional features:

1. hybrid development technology server side scripts,
2. partial caching of dynamic content,
3. transparent service state migration.

Prototype environment will be constructed to demonstrate the above features and a set of comprehensive experiments will be performed. The comprehensive set of experiments should cover all aspects of the new features. They should cover the following:

1. Demonstrate speed of data transfer compared to the standard Apache speed
2. Demonstrate speed of dynamic content and partial content retrieval capability of the proposed system versus standard Apache speed
3. Demonstrate maximum number of concurrent connections and throughput versus what is provided by standard Apache
4. Demonstrate Hybrid C++, Java Technology service capability.
5. Demonstrate delay parallelization
6. Demonstrate caching and buffering
7. Demonstrate the service state migration
8. Demonstrate containers/ channels pooling
9. Demonstrate server page caching
10. Demonstrate Stress benchmark testing using different realistic workload mix versus other standard servers
4. Work Done Before Thesis Proposal

4.1. Design

An Object Oriented UML design has been developed for the container and the HPA. Basic classes were identified and a class diagram was constructed, as well as use case scenarios, action flow diagrams, data flow diagrams, and deployment diagrams. The Visual Paradigm CASE tool was used to develop the first design draft.

4.2. Prototyping and Implementation

A C++ prototype for the container with the basic functionalities is available and has been used for testing purposes with without Cluster Management System, Deployment Manager, and Discovery Service that are still not implemented. Also, a prototype for the HPA is available without the skeleton cache and the discovery client. Basically, what is finished is a communication layer prototype for both the HPA and C++ Container, a C++ Container Service Factory, a C++ Container Service Dispatcher, and C++ Configuration Manager.

4.3. Experiments and Results

A set of experiments have been conducted on the available prototype as means of probing the water to see how promising the proposed architecture is and to identify also the prospect overheads and bottle necks as early as possible. Those set of experiments are considered a proof of concept experiments designed to evaluate the new architecture at it's premature stage.

Two sets of experiments were carried out, the first set is designed to apply an exhaustive brute force test to get a feeling of performance with respect to scalable stress using a single channel setup compared to Apache web server. A lot of available web servers are based on Apache such as IBM HTTP Server and Oracle HTTP Server. The same experiment is performed using different work loads defined by the number of clients and results are collected for each run and compared against Apache Web Server. By this we will define the minimum performance gain that we can achieve using UDP over normal TCP used by traditional web environments. The second set of experiments is designed to show the benefits of the multichannel setup with different incremental work loads together with processing activities/delays on the server side; the objective of this set of experiments is to show the effect of the proposed multi-channel server pages technology, and how processing intensive pages will benefit from it.
4.3.1. Single Channel Exhaustive Experiment

**Environment:**

<table>
<thead>
<tr>
<th></th>
<th>Apache</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Servers</td>
<td>1 Apache Server</td>
<td>1 Container Server</td>
</tr>
<tr>
<td># of HPAs</td>
<td>Does not apply</td>
<td>1 HPA located on the client side</td>
</tr>
<tr>
<td>Network Speed</td>
<td>100 Mb/Sec</td>
<td></td>
</tr>
<tr>
<td>Size of Traffic transferred</td>
<td>100 MB</td>
<td></td>
</tr>
<tr>
<td>Traffic Generator</td>
<td>A traffic generator based on the CURL library to generate concurrent traffic</td>
<td></td>
</tr>
<tr>
<td>Number of Total Client Connections</td>
<td>Varies from 100 to 112 Clients connection according to each run setup</td>
<td></td>
</tr>
<tr>
<td>Number of Concurrent Clients</td>
<td>Ranges from 2 concurrent clients till 25 concurrent clients, with increment of 2 concurrent clients connections.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

In these experiments we will perform 11 runs, each run will have a number of concurrent clients. The number of client connections are designed such that no run will have less than 100 client connections.

With the help of the Traffic Generator, batches of concurrent clients are dispatched to be served in parallel, applying different stress levels on each web environment.

A 1 second delay is applied by the generator between each concurrent connection batch.

Although this set of experiments is designed for the single channel setup, nevertheless we have introduced in the test the results of applying the same runs to multi-channel environments on single server, to show the limitation of the multi-channeling without hardware clustering.

**Test Aspects**

1. Test Speed of data transfer
2. Test Maximum number of connection with respect to time.
3. Test Traffic fluctuation
Results:

Average Time (Sec)

Max Client Service Time (Sec)

Average Client Bandwidth (MB/sec)

Maximum Client Bandwidth (MB/Sec)

Minimum Client Bandwidth (MB/Sec)

Fluctuation Time (Sec)

Fluctuation Bandwidth (MB/Sec)

Total Bandwidth (MB/sec)

Total Experiment Duration (Sec)
### 4.3.2. Multichannel Delay Parallelization Experiments

#### Environment:

<table>
<thead>
<tr>
<th></th>
<th>Apache</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Servers</td>
<td>2 Apache Server</td>
<td>2 Container Server</td>
</tr>
<tr>
<td># of HPAs</td>
<td>Does not apply</td>
<td>1 HPA located on the client side</td>
</tr>
<tr>
<td># of Channels</td>
<td>Does not apply</td>
<td>2 Channels</td>
</tr>
<tr>
<td>Layer-7 Dispatcher</td>
<td>1 Round-Robin Dispatcher</td>
<td>Does not Apply</td>
</tr>
<tr>
<td>Network Speed</td>
<td>100 Mb/Sec</td>
<td></td>
</tr>
<tr>
<td>Size of Traffic transferred</td>
<td>20 MB</td>
<td></td>
</tr>
<tr>
<td>Traffic Generator</td>
<td>A traffic generator based on the CURL library to generate concurrent traffic</td>
<td></td>
</tr>
<tr>
<td>Number of Total Client Connections</td>
<td>Varies from 100 to 112 Clients connection according to each run setup</td>
<td></td>
</tr>
<tr>
<td>Number of Concurrent Clients</td>
<td>Ranges from 2 concurrent clients till 14 concurrent clients, with increments of 2 concurrent connection.</td>
<td></td>
</tr>
<tr>
<td>Processing Delay</td>
<td>Ranges from 0 to 16 seconds processing delay, with increment of 4 seconds of delay.</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>In these experiments we will perform 7 runs for each delay value resulting in 35 total runs. Each run will have a number</td>
<td></td>
</tr>
</tbody>
</table>
of concurrent clients and processing delay. The number of client connections are designed such that no run will have less than 100 client connections.

With the help of the Traffic Generator, batches of concurrent clients are dispatched to be served in parallel, applying different stress levels on each web environment.

A 1 second delay is applied by the generator between each concurrent connection batch.

In the case of the Apache Server, a single service will be fetching data and sending it to the browser, different processing delay values will take place before sending the data in each run. In the case of the container, a skeleton service is designed to split the data into two halves and retrieve each half concurrently, and in this case the same delay will be distributed over the 2 channels.

| Test Aspects | 1. Test Speed of data transfer with respect to the increase in processing delay.  
| Test Aspects | 2. Test Maximum number of connections with respect to time relative to the increase in processing delay.  
| Test Aspects | 3. Test Traffic fluctuation with respect to the increase in processing delay. |

**Results:**

0 seconds processing delay:
Figure 3: Multichannel With 0 seconds delay
4 seconds processing delay:

[Graphs showing average, maximum, and minimum time and bandwidth for Apache and Container with different numbers of clients]
Figure 4: Multichannel With 4 seconds delay

8 seconds processing delay:
Figure 5: Multichannel With 8 seconds delay
12 seconds processing delay:

- Average Time (Sec)
- Maximum Time (Sec)
- Minimum Time (Sec)
- Average Bandwidth (MB/Sec)
- Maximum Bandwidth (MB/Sec)
- Minimum Bandwidth (MB/Sec)
- Fluctuations Time (Sec)
- Fluctuations Bandwidth
- Total Bandwidth (MB/Sec)
Figure 6: Multichannel With 12 seconds delay

16 seconds processing delay:
Figure 7: Multichannel With 16 seconds delay
Bandwidth Gain Relative to increase in Processing Delay

% in Bandwidth Gain Per Client

2 Clients

% in Bandwidth Gain Per Client

4 Clients

% in Bandwidth Gain Per Client

6 Clients

% in Bandwidth Gain Per Client

8 Clients

Bandwidth Gain %

0
10
20
30
40
50
60
70
80
90
100

Bandwidth Gain %

0
10
20
30
40
50
60

Bandwidth Gain %

0
5
10
15
20
25
30
35

Bandwidth Gain %

0
5
10
15
20
25
30
35
Figure 8: Bandwidth gain relative to increase in processing delay.
4.3.3. Observations and Result Analysis

1. Single Channel UDP is faster than TCP even with the existence of the HPA over head.
2. In the single channel experiments the container can handle the same amount of connections as Apache in less amount of time.
3. The fluctuation of the UDP communication is much higher than the TCP.
4. In multichannel experiments, as the processing time increases, the gain in the speed increases.
5. In multichannel experiments, very poor gain in performance is achieved in the case of no processing delay, so it is better to use single channel when there is no delay that can be parallelized. This is basically due to the overhead of handling more connections which is of no value in case of intensive pure data transfer applications. Thus, applications that have interlacing I/O and processing would be the best candidate for multichannel environment, e.g. Web application based on database transactions.
6. In multichannel experiments, no skeleton caching is provided. Better performance is expected when skeleton caching is enabled.
7. In multichannel experiments, higher overall bandwidth is achieved in case of the UDP container over Apache.
8. The experiments setup is favoring apache as the number of HPAs in all the experiments is one which is an overhead, as in the case of single channel setup a web transaction is served through 2 network connections by the HPA in comparison with one network connection in the case of apache. And in the multi channel case it is even worse, as a single web transaction is served by apache using 2 network connections (1 connection for dispatching + 1 connection for the service), while in the case of the multi-channel environment a web transaction is served through N+2 network connections where N is the number of channels and the other 2 channels are one for the fetching of the skeleton (or checking if not updated) and the other is the TCP web client connection; Hence if the experiments are applied from different clients, we expect the results to be better.
9. The fluctuation observed by the results graphs is due to the large number of threads required on the HPA side to serve a web transaction especially in the multichannel. In real life case, each HPA will be serving one web browser, which will be of less load on the HPA and thus the fluctuation in serving clients will be expected to be less. Also, some tuning of the pthread library priority initialization parameters may need to be investigated to make sure that all the HPA clients have close execution CPU slices.
10. As the HPA with its TCP reverse proxy connection is an over head, better performance can be achieved if the HPA is integrated in the web client.
5. Work Done So Far

5.1. Design

The rest of the design is completed and enhanced. Mainly design diagrams related to the Custer Management Subsystem and the Deployment Manager are completed and detailed depth in the design is achieved as feedback mechanism from implementation is adopted iteratively and continuously.

Moderate design updates, changes, and enhancements are expected as the prototype is approaching its final shape, yet most of the work on this front is done.

5.2. Prototyping and Implementation

1. The C++ Container is finalized and a lot of bugs are identified and fixed.
2. The Java Container version is fully implemented.
3. Container Cluster Management Sub System, for both C++ and Java Containers, are implemented and integrated into the container code including Service Discovery. (Some subsystems within the cluster management Subsystem are postponed and will be presented in the next section)
4. HPA Channel Reservation module.
5. HPA Discovery Service.
6. Remaining Tasks and Time line

6.1. Remaining Tasks

1. Complete the missing parts of the Cluster Management
   a) Container Nodes Statistical Data Replication
   b) APIs needed by the deployment manager to be able to deploy services
2. Deployment Manager
3. HPA Skeleton Cache.
4. Comprehensive testing for the whole system after being put together.
5. Constructing experiments and test cases.
6. Building the Clustered Environments for the experiments.
7. Running Experiments and System Tuning based on the results.

6.2. Time line

<table>
<thead>
<tr>
<th>Task</th>
<th>From</th>
<th>To</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing Parts of Cluster Management</td>
<td>1/12/2008</td>
<td>9/12/2008</td>
<td>7 Days</td>
</tr>
<tr>
<td>Deployment Manager</td>
<td>10/12/2008</td>
<td>20/1/2009</td>
<td>30 Days</td>
</tr>
<tr>
<td>HPA Skeleton Cache</td>
<td>21/1/2009</td>
<td>10/2/2009</td>
<td>15 Days</td>
</tr>
</tbody>
</table>