

# Project Report for Comp520

## *A Web Server Architecture*

### *for*

## *Symmetric Multiprocessor System*

Santashil PalChaudhuri (*santa@rice.edu*)  
Rajnish Kumar (*rajnish@rice.edu*),  
Amit Kumar Saha (*amsaha@rice.edu*),

December 17, 2000

### Abstract

We propose a **Co-AMPED** (*Coordinated Asymmetric Multi Process Event Driven*) architecture for a web server executing on a Symmetric Multiprocessor (**SMP**) system. For a  $k$  - processor SMP system we execute  $k$  copies of an **AMPED** process which *coordinate* among themselves to optimize the usage of the cache across the  $k$  processes. We compared the performance of the proposed model against a Multi Process (**MP**) architecture and found that it performed 50% better on an average. However we found that there is no improvement by increasing the number of main processes in any of the architectures considered. Instead the **AMPED** implementation without any modification was performing best. For our analysis we used the Flash <sup>1</sup> version of **MP** and made changes to the Flash implementation of **AMPED** to get an implementation of the **Co-AMPED** architecture.

## 1 Introduction

The internet is fast becoming *the* primary source of information exchange. the speed at which requests need to be handled by a web server is becoming critically important as compared to other factors such as available bandwidth. the communication links are becoming faster. thus the issue of bandwidth utilization becomes very crucial. Typically there are three ways in which present web sites handle huge amount of Internet traffic

- web replication (mirroring)
- distributed caching
- server performance enhancement

nowadays when the cost of installing and maintaining a  $k$  - processor system is significantly less than having  $k$  separate single processor system (as in web mirroring), we need to design

---

<sup>1</sup>Copyright (c) '1999' RICE UNIVERSITY. All rights reserved. Created by Vivek Sadananda Pai [vivek@cs.rice.edu], Departments of Electrical and Computer Engineering and of Computer Science.

our web servers so that they can use the multiprocessing capability of these  $k$  - processor machines in a more efficient manner.

We stress on the last option. We try to find out the best possible approach that can be taken to enhance the performance of a web server running on a  $k$  - processor SMP system. In particular we compare the *multi process (MP)* model to the *asymmetric multi process event driven AMPED* model.

Web servers take different approaches towards enhancing their performance. The *single process event driven (SPED)* architecture performs quite well for workloads where most of the requested content is in main memory. Zeus server[3] and Squid caches are examples of this.

When the workload is too big to fit into the main memory *multi process (MP)* or *multi thread (MT)* architectures usually perform quite well. Apache[1] uses MP and MT architectures on Unix and Windows NT respectively.

The *asymmetric multi process event driven (AMPED)* architecture performs best under both cached workloads as well as disk intensive workloads. Flash[2] is an implementation of an AMPED architecture.

However none of these models have any optimizations for executing on a *symmetric multi processor* system. This paper presents an extension to the AMPED architecture called *coordinated AMPED (Co-AMPED)* since here there are multiple AMPED processes coordinating among themselves.

The rest of this paper is organized as follows. Section 2 explains the compared architectures. The design and implementation of Co-AMPED is explained in Section 3. Section 4 contains the performance evaluation of the MP and Co-AMPED model. Section 5 has some related work in this field.

## 2 The compared Architectures

Server architectures normally fall into one of the following categories :

- Multi Process (**MP**)
- Multi Threaded (**MT**)
- Single Process Event Driven (**SPED**)
- Asymmetric Multi Process Event Driven (**AMPED**)

We however compare only the web server architectures explained below. We did not go for the MT model because the basic concurrency issues remain the same in both the MP and the MT model but the MT model is practically more demanding to code. So we chose to compare the MP model to the Co-AMPED model. However if we were to choose the MT model then we would have compared it to a coordinated multi *threaded* event driven (**Co-AMTED**) architecture.

### 2.1 Multi Process

In the *multi-process* (MP) architecture, a process is assigned to execute the basic steps associated with serving a client request sequentially. The process performs all the steps related

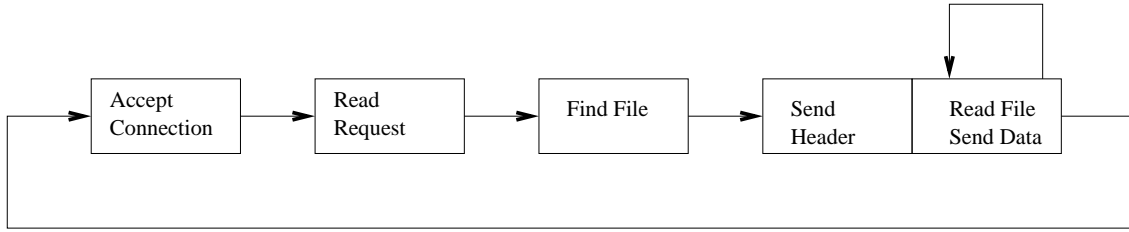


Figure 1: MP Architecture

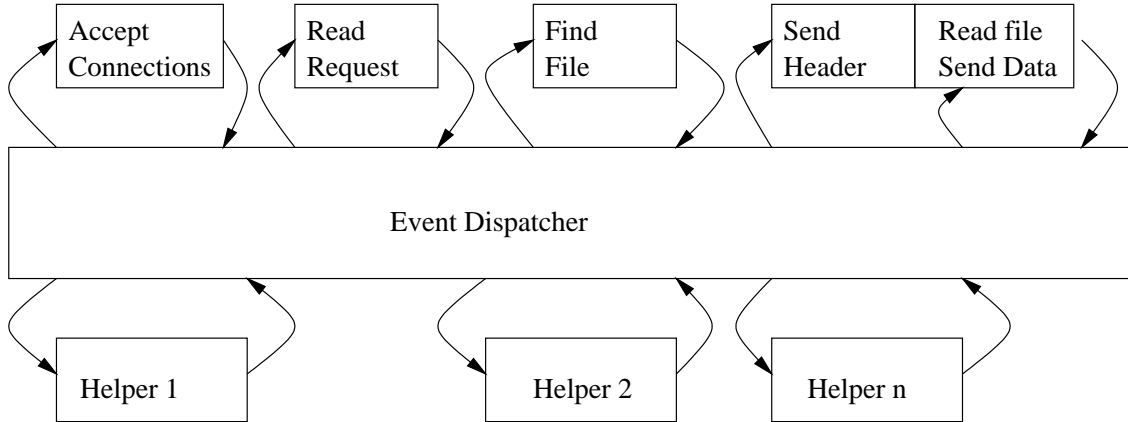


Figure 2: AMPED Architecture

to one HTTP request before it accepts a new request. Since multiple processes are employed (typically 20-200), many HTTP requests can be served concurrently. Overlapping of disk activity, CPU processing and network connectivity occurs naturally, because the operating system switches to a runnable process whenever the currently active process blocks. Since each process has its own private address space no synchronization is necessary to handle the processing of different HTTP requests. However, since no information is shared it might be more difficult to do any optimizations based on global information, such as shared cache of valid URLs. Figure 1 shows the architecture.

The introduction of  $k$  - processors gives it the ability to spawn more processes and hence to handle more requests.

## 2.2 Coordinated Asymmetric Multi Process Event Driven

An event driven architecture can be thought of as a state machine that performs one basic step associated with serving an HTTP request at a time, interleaving the processing steps associated with many HTTP requests. In each iteration the server does a **select** to check for completed I/O events.

The Asymmetric Multi-Process Event Driven (AMPED) architecture combines the event driven approach of the event driven architecture with multiple *helper* processes that handle blocking disk I/O operations. When a disk operation is necessary the main server process instructs a *helper* process via an IPC channel to perform a potentially blocking operation. Once the operation completes, the helper process notifies the main process, again via an IPC channel. Figure 2 shows the architecture.

With a  $k$  - processor system there can be multiple such main processes running, each handling requests as per the AMPED model. There is however no coordination among the individual processes.

The *coordinated* AMPED (Co-AMPED) model runs  $k$  AMPED processes but these processes coordinate among themselves to share *global information* in order to implement certain optimizations.

### 3 Design and Implementation

We chose Flash for the implementation of Co-AMPED because it already has the AMPED architecture implemented. It would be much easier to simply extend it to get an implementation for Co-AMPED than to write it afresh. Besides Flash also has an MP version, which we could use for our comparisons.

The following were the major design issues and how we chose to implement them.

- Our Co-AMPED model or our MP model would have  $k$  copies running on a  $k$  processor SMP machine. However we could have chosen to run any number of copies to run on the SMP system. We chose the number of *main processes* to match the number of *processors* because of the intuition that the operating system would be able to have a one-to-one mapping between the processes that it has to run and the processors. Here we assume that the system on which the processes are running is a dedicated web server and hence there are very few other jobs (daemons anyway run very infrequently) that need to be frequently scheduled. This might lead to an *affinity* between a process and a processor. Ideally this would decrease the context switching overhead.
- Next we had to decide on the information that we wanted to share among the  $k$  processes. We chose to maintain a hash table of URLs with each URL having an integer associated with it which indicates the main process (among the  $k$  different main processes) that last serviced a request for this URL. We decided to maintain this information in a *shared memory* data structure so that it would be accessible to all the main processes.
- We also needed to decide on a policy for using this shared information. When one among the  $k$  main processes receives a request from a client it does the following :
  1. Get the URL that has been requested.
  2. Query into the shared information to find if there exists another process which had handled it. (i.e does that URL in the hash table have an associated process number)
  3. If there exists such a process then send the request to that process and delete that request from the queue of requests.
  4. If there exists no such request then process the request as would have been done in the normal AMPED model.
  5. If the process has executed step 4 the shared information corresponding to the URL serviced needs to be updated with its process number.

Any one of the  $k$  main processes, apart from receiving a request directly from the client could also potentially receive a request from any of the other main processes. For that it goes through the following steps :

1. Receive a request from some other main process.
2. Handle the request as any other normal request.
3. Update the global information corresponding to the URL of the request that it received from another process.

The request is forwarded because we want the caches of the processors to have as different values as possible so that we manage to cover as much of the *working set* of files as possible. We could have also designed such that each process on receiving a request *first* checks whether the file requested is present in the main memory and if so then handles the request, else sends it to another process. However in that case the cache entries may get replicated. Our policy tries to keep this replication to a minimum. Besides retrieval from cache would be faster than retrieval from main memory. However we have to pay with the price of sending the request.

- If we only look at the URL hash table and decide to forward a request for a particular URL to some process then a number of consecutive requests for the same file would throttle that process which had been handling that request, while the other processes would become idle (after forwarding the request). This would lead to loss in performance rather than any gain.

However we have not been able to implement any load function as such and we rely on the unrealistic assumption that there will be an even distribution of the requested files.

- Some care has to be taken when forwarding a request to another process because the socket descriptor identifying the connection to the client which originally sent the request must still be valid in the new process ( to which the request has been forwarded to ). We used `socket()` with *host internal protocol* and `sendmsg()` to pass the socket descriptor as well as other information of an HTTP connection from the client.
- In order to prevent the repeated forwarding of a request we allow only one forwarding of a request.

## 4 Results

Here we present the results of the experiments we performed to evaluate the Co-AMPED model against the MP as well as AMPED architectures. We compare the performance between these models for

- a set of clients repeatedly requesting the same file, where the file size is varied in each test.
- varying workload with a range of file sizes, generated synthetically.

We also show the effect of varying the number of main processes for the Co-AMPED and the AMPED models, on the performance of the server.

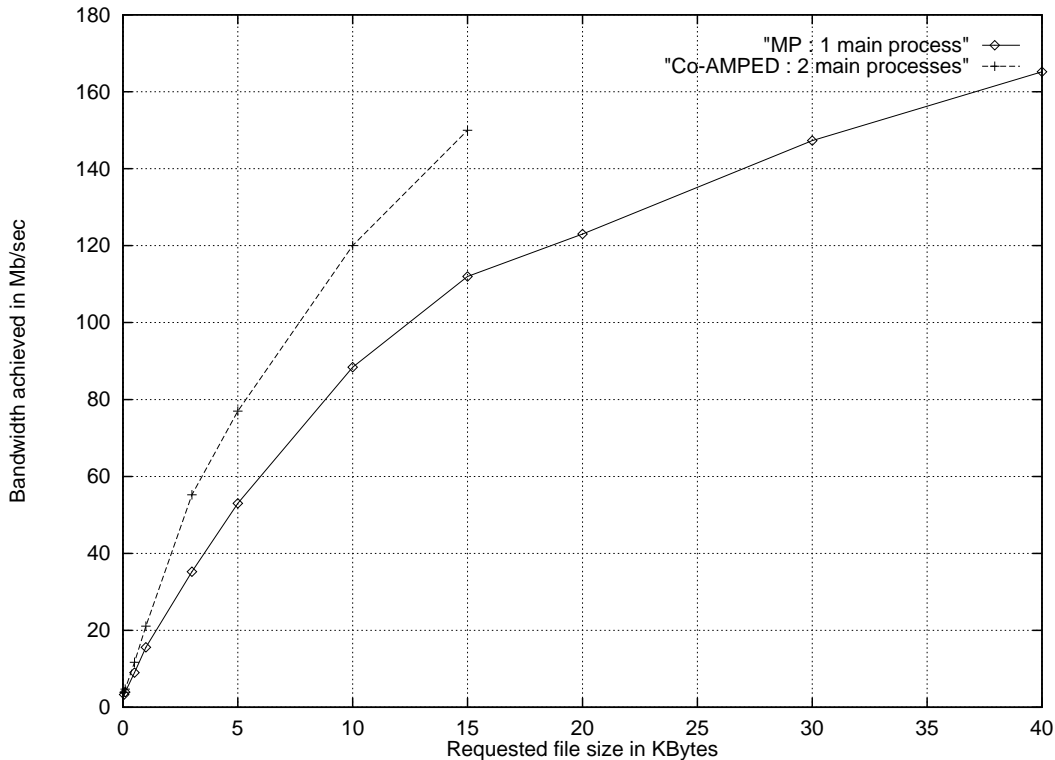


Figure 3: Comparison of MP and Co-AMPED implementation of Flash

## 4.1 Experimental Setup

The experiments were performed with the clients running Free BSD 2.2.6 on 800MHz AMD Athlon(tm) processors. All tests use the same server hardware based on four 500 MHz Pentium III CPU with 1 GB of memory and multiple 100 Mbits/s Ethernet interfaces, and running Free BSD 4.0. A switched Fast Ethernet connects the client machine and the server machine. The client software is an even-driven program that simulated multiple HTTP clients. Each simulated HTTP client makes HTTP requests as fast as the server can handle them. The clients showed about 50% CPU load (under all test loads), which means that the clients were indeed able to issue requests as fast as the server could respond to them.

## 4.2 Performance Comparison

### 4.2.1 Single file workload

In this experiment a set of clients repeatedly request the same file, where the file size is varied in each test. This is an unrealistic and simple workload that allows the servers to perform at their best since the file can be cached by the server. The proposed architecture did perform about 50% better than the naive MP model. The results are shown in Figure 3. This is expected since the Co-AMPED model does not have context switching overhead unlike the MP model. For smaller files, the bandwidth achieved is less because the parsing of the URL takes a substantial amount of time compared to actual data transfer.

### 4.2.2 Varying workload

The workload, in this case, consists of a set of files of varying size. The frequency of request for the files also varies so that it *resembles* a realistic workload in which some files are invariably requested more often than others. The following were the file distributions that were used

- Workload 1 : This included two files of sizes 500Bytes and 1KByte, each being requested with equal probability.
- Workload 2 : This included three files of sizes 500Bytes, 1KByte and 5KByte each being requested with equal probability.
- Workload 3 : This included 4 files of sizes 50Bytes, 500Bytes, 1KByte and 5KByte each being requested with equal probability.
- Workload 4 : This included 10 files of which files of sizes 50Bytes, 500Bytes, 1KByte, 3KByte, 5KByte, 10KByte, 15KByte, 20KByte are requested with equal probability. A file of 50KBytes was requested with half the probability while a file of 100KByte was requested with one tenth the probability.

The comparison is shown in Figure 9. One important thing to note is that the machine that we tested on had a 1 GB memory and hence even for a realistic workload things behaved like the single file workload since all the working set files were getting accommodated in the main memory itself and there was not much disk access.

### 4.2.3 Scaling with the number of main processes

For a single file workload, we vary the number of main processes for the MP,AMPED and Co-AMPED architecture and measure the performance. The results are shown in Figure 4, Figure 5 and Figure 6.

We observed that performance of all the models except MP decreased as the number of main processes was increased. In case of MP model, this can be explained because increase in the number of main processes allows more number of processes to be spawned. The particular implementation of MP limits the number of processes that a main process can spawn to 32. Increase in number of processes does lead to more context switching overhead, but for a heavy workload that is overshadowed by the increase in computational power. Presence of more processes leaves less memory for files, leading to more memory misses. However since main memory size is quite large here, we do not see that having any effect here.

We expected the performance to increase in case of Co-AMPED and AMPED models with increase in number of main processes. But as Figure 5 and Figure 6 shows, performance actually decreased. This is probably due to some serialization problem in the Flash implementation of AMPED. Some part of the code is inherently serial and from the performance results it seems that this part dictates the processing of a particular request. The workload on the CPU does increase proportional to the number of main processes running, but the bandwidth achieved decreases, which is due to the context switching and memory utilization of the processes. Server over-utilization is ruled out since the server is loaded to a maximum of 60% - 70% of full load. There might be serializability issues in the underlying OS as well, but we have not analyzed that as yet.

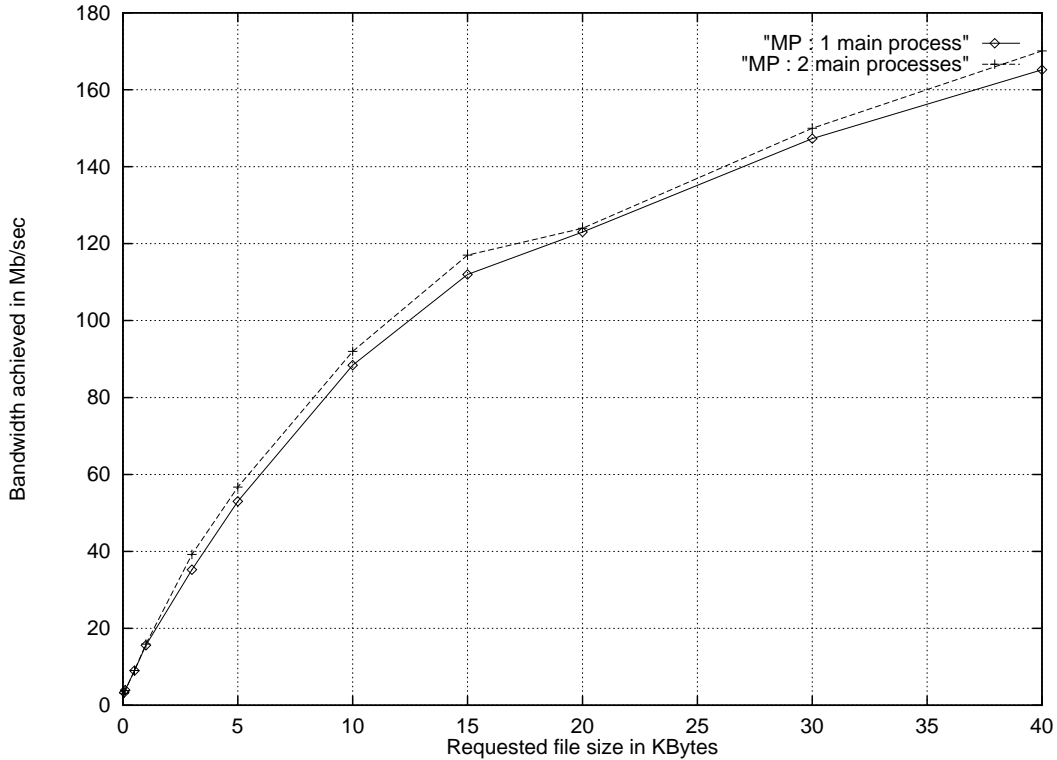


Figure 4: Scaling of MP implementation of Flash

Due to this problem of scaling the performance of Co-AMPED version of Flash with 2 main processes is less as compared to the AMPED version with 1 main process. Since the architecture does not scale hence the best performance for the proposed architecture is with 2 main processes.

#### 4.2.4 Overhead of Co-AMPED implementation of Flash

Since our implementation of the Co-AMPED architecture uses almost no optimization, hence there is some overhead as compared to the basic AMPED model which runs with a single main process. The overhead is shown in Figure 7. This overhead can be done away with by coding in a more efficient manner.

## 5 Related Work

Co-AMPED is motivated from AMPED architecture of the Flash web server. Flash was proved to run considerably faster than other architectures, like Single Process Event Driven (SPED) and MP, on single processor systems. But AMPED was not optimally scalable for multi-processor systems. Co-AMPED is an extension of AMPED model to make it fully scalable and optimal for SMPs. Request scheduling and load balancing across different main processes and processors roots from the similar issues involved in designing web server architecture for clusters. The most common approach behind request scheduling in web server running on cluster has been to have a devoted central process, running possibly on a devoted processor, which receives all requests, and distributes them across other processes

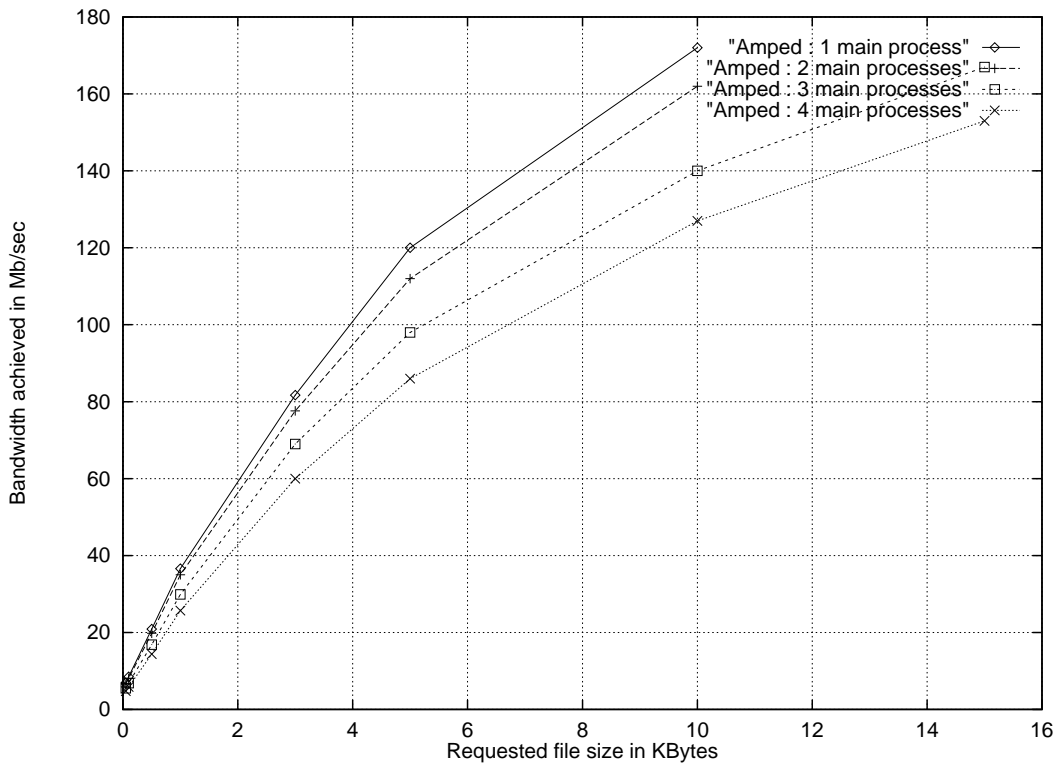


Figure 5: Scaling of AMPED implementation of Flash

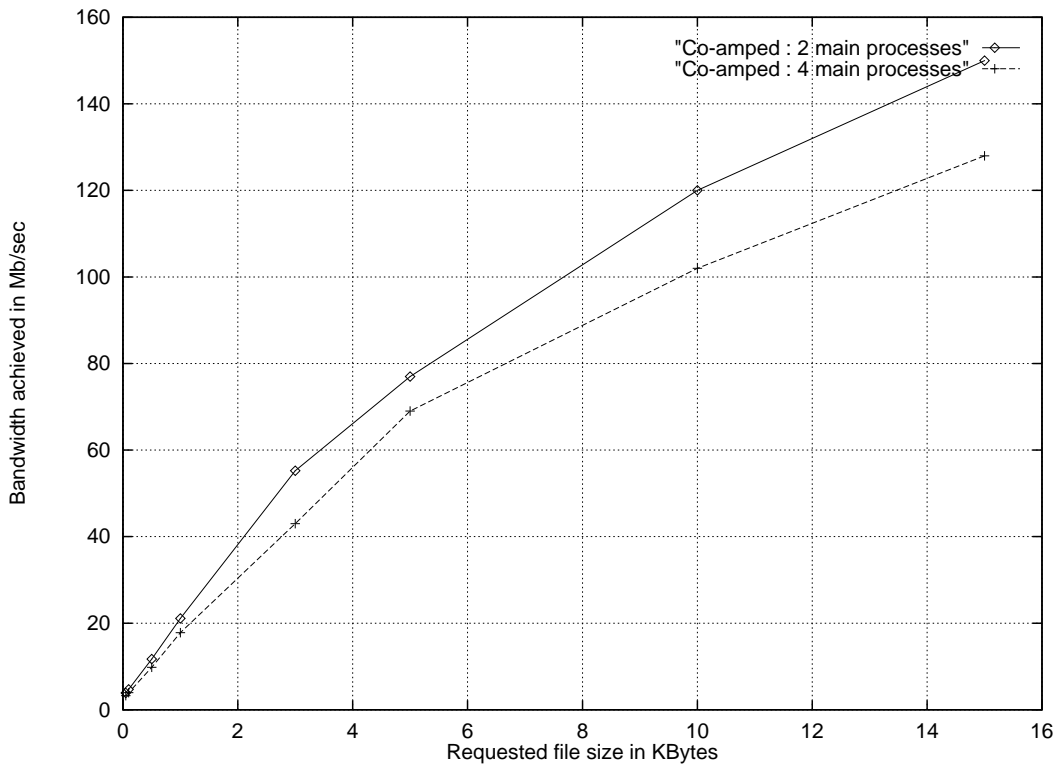


Figure 6: Scaling of Co-AMPED implementation of Flash

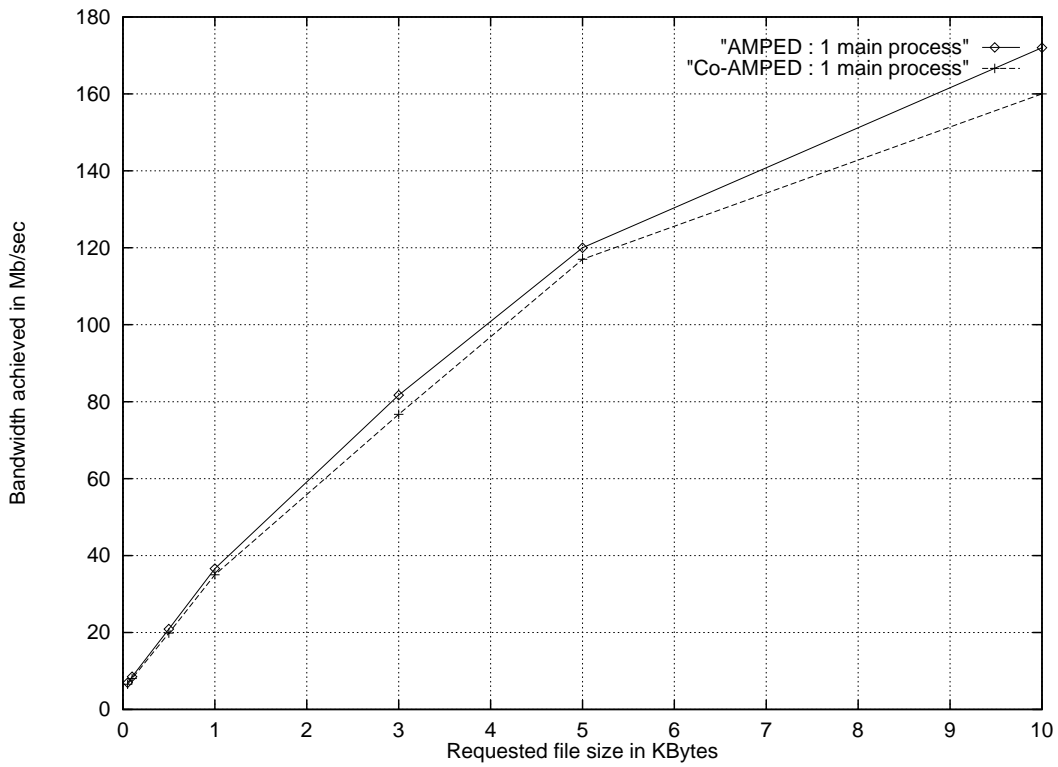


Figure 7: Overhead of Co-AMPED implementation of Flash

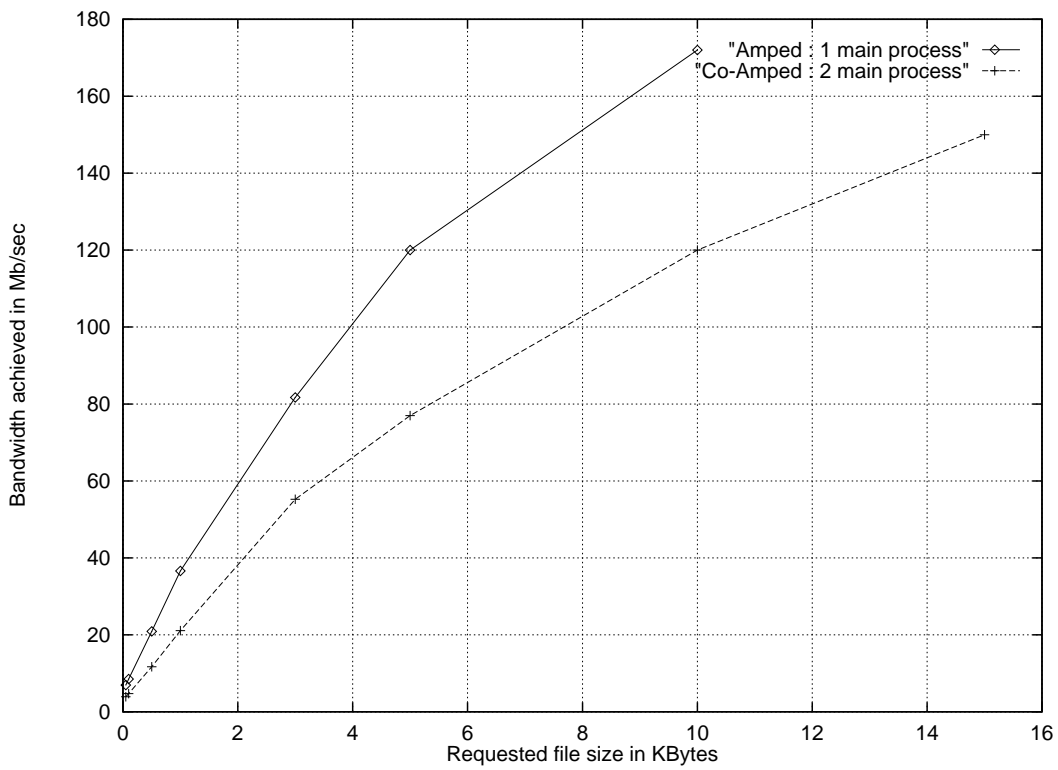


Figure 8: Comparison of AMPED and Co-AMPED implementations of Flash

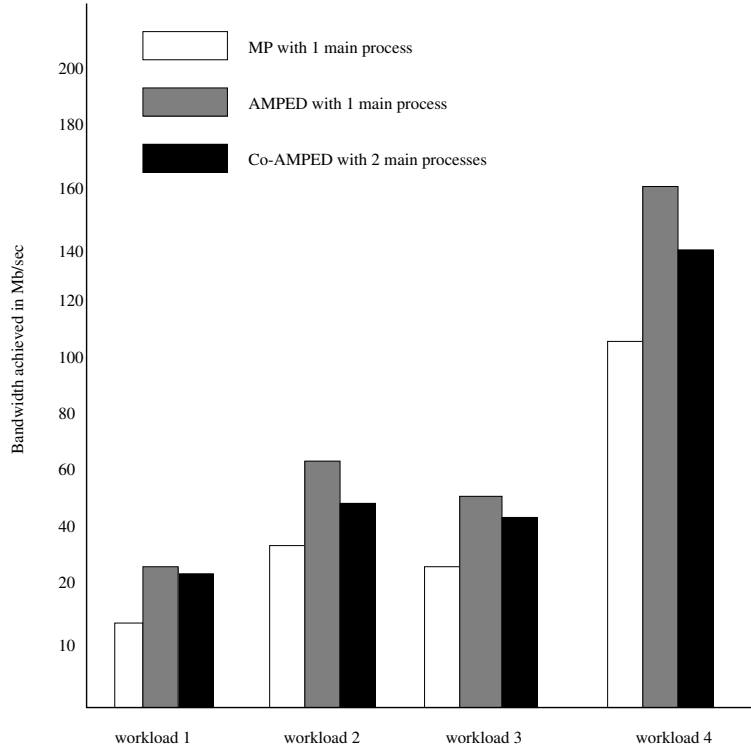


Figure 9: Performance comparison under realistic workload

based upon locality awareness ( LARD ) [4]. Implementation of Co-AMPED doesn't need any such devoted process or processor.

## 6 Future Work

A web server based upon our proposed architecture, Co-AMPED, performs 50% better than that based upon MP model. The performance gain is achieved because of less context switching overhead and better use of data lying in the cache of the processors. Cache data utilization is gained by scheduling the request across different main processes running on different processors. Request scheduling is done based upon past history about which main processes handled the requested URL.

As part of future work, context switching overhead of Co-AMPED may further be brought down by using threads instead of processes. The helper processes can be replaced by helper threads and possibly even main processes can be replaced by main threads. Also, instead of forwarding the request based only upon the past history, the load on a particular main process could be considered before forwarding any request to that process. This load might take into account parameters like the size of the file requested, because for small files, the forwarding overhead may be higher than serving the request directly even though it is not in the cache.

Better performance measurement would help us in understanding the characteristic of a working set that would make the Co-AMPED approach most efficient. This could again be used to decide whether or not to forward requests. We would like to request files available

from traces from actual web sites so that we get a more realistic estimate of the performance of the implementation.

As of now CGI script handling has not been modified to implement any kind of *coordination*. We would like to do that.

However as the performance results prove the AMPED implementation of Flash is not scalable. Since we implemented Co-AMPED over the already implemented AMPED version of Flash, the Co-AMPED implementation was also not scalable with the number of main processes. We plan to identify the bottlenecks of the AMPED implementation and that would allow us to make the Co-AMPED model scalable as well.

We hope to address these issues in the near future.

## 7 Acknowledgments

We are grateful to Sitaram Iyer for his valuable comments and help.

## References

- [1] Apache; <http://www.apache.org>
- [2] Vivek Pai, Peter Druschel, Willy Zwaenepoel; *Flash: An efficient and portable Web Server*; Proc. of the 1999 Annual Usenix Technical Conference, Monterey, CA, June 1999
- [3] Zeus Technology Limited; *Zeus Web Server*; <http://www.zeus.co.uk>
- [4] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel and Erich Nahum; *Locality-Aware Request Distribution in Cluster-based Network Servers*;
- [5] MESI Cache Coherency Protocol <http://www.lintech.org/CE302/MESI.html>