Parallel Computing with ParaComputeD Server

Vishal Singh, Sifat Shaikh, Urvi Chawla

Abstract— The purpose of this project is to reduce the time required for processing and execution of a program by developing a framework that will basically delegate the work to various servers best suited for that particular task. We propose a framework that will reduce the processing time basically by the principle of parallel computing by using the concept of "Job Delegation". This framework is basically a job delegation server which will delegate the work to available node/workers. This allows for non-blocking parallel processing.

Index Terms—Parallel Computing, ParaComputeD, Job delegation server, Load Balancing, Bandwidth, Constant polling, Workers.

1 Introduction

PARALLEL computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved at the same time.

In the existing system, as shown in the diagram (fig. 1.1.) below of a typical form submission example, we can see that each step has to wait for the previous step to be completed. And the user has to wait till all the processes are completed.

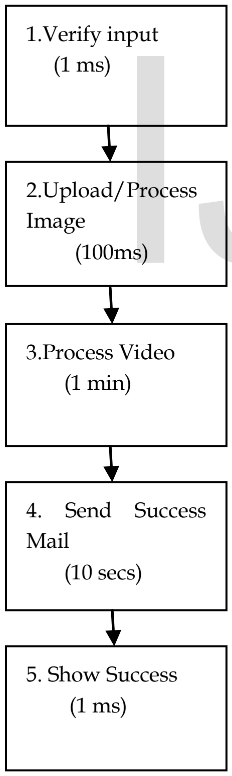

Fig. 1.1: Typical Form Submission Example

In the above example as you can see al the process are being done in separate steps but not simultaneously. Thus, the processes have to wait for the previous process to be completed for its execution to be started. Hence the time required for completion of the whole task sums up the time required for each individual process. This results in increasing the overall processing time. The proposed system avoids this issue by simultaneously processing the separate processes, thus, considerably reducing the processing time.

In parallel computing, a computational task is typically broken down in several, often many, very similar subtasks that can be processed independently and whose results are combined afterwards, upon completion.

2 The Existing System

In order to have an efficient web application, it should have more reliable and robust code, proper security, intelligent routing mechanism for quick response and zero downtime during upgrade process.

In the present system, a queuing system which is a generic model that comprises of three elements:
A user source, a queue and a service facility that contains one or more identical servers in parallel. Each user of the queueing systempasses through the queue where he may remain for a period of time and then is processed by a single server because of the parallel arrangement of servers. Once a user has left the server, after obtaining the service, the user is considered to have left the queue using system as well.

A queuing system is formed from three generic elements:
1. The arrival process of users in the system;
2. The order in which the users obtain access to the service facility, once they join the queue;
3. The service process.

The present system on which parallel computing is based is noraml queues. These queues used have many problems. Most of them can be resolved by proposed framework.

The major problems are : Problem of waiting, Constant polling, Balancing of load.

In case of queues which are dependent on others to obtain necessary information or data in order to pack it into a single entity or component. The queue which needs to obtain the desired information constantly needs to poll the other queue

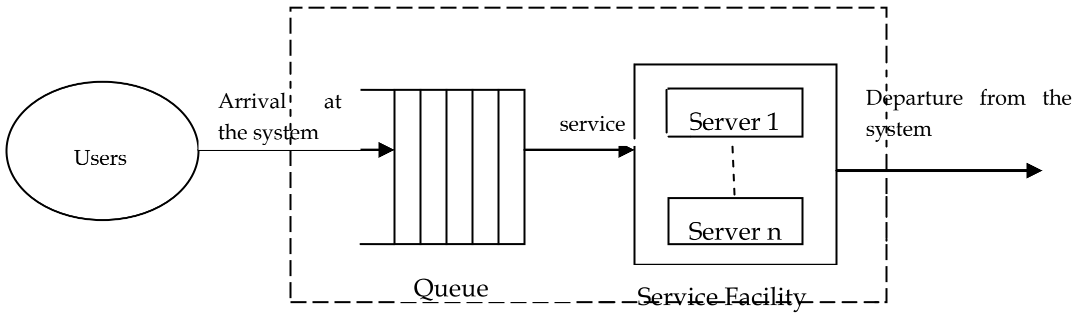

Fig 2.1 : Queuing System

.
This leads to a wastage of bandwidth which in turn blocks the channel and reduces throughput. Also the processes need to constantly poll the queues to check for availability of work.

Finally arises the major problem of balancing of load. In the present system all the workload is put into the queue which is closest to that particular server. Server basically keeps pushing the entire load into the nearest queue. This makes that particular queue full and the process overloaded. This in turn affects the overall processing and can lead to a system crash.

3 Proposed System

We have proposed a ParaComputeD Server as the solution to the proposed problem of the current parallel processing system. ParaComputeD is a job delegation server. It is built for Robust messaging between applications. It has built-in support for load-balancing and fault tolerance. It can be used to build distributed applications that scale right from the start and build applications that have no single point of failure as well as decouple your program into components written in languages best suited for the task.

This framework resolves the problem of witing by allocating the desired task to desired process at the desired time, resulting in no queue and process overloading. This is very essential as else it leads to unnecessary delays and wastage of resources of computation.

Constant polling is another issue which is overcome by using this framework. This framework allocates the workload to the respective queues as and when the work arises. This ensures that efficiency of usuage of bandwidth is maintained. Thus, the need of a process to constantly poll an queue is eliminated.

The most important issue that is resolved by the ParaComputeD framework is the issue of balancing of load.This balancing of load is basically the division of the entire workload, to all the processes and queues that exist to help the system function. Thus, all the processes irrespecctive of their location and distance from the main server are alloted work depending on the workload.

ParaComputeD is built to solve the problem of existing queuing softwares which were either proprietary or bulky or slow. The present scenario desires something fast, lightweight and well documented, which would be as reliable & robust as any enterprise software.

Another goal of ParaComputeD is to keep things simple. ParaComputeD uses a simple protocol based on websockets. This makes it as easy to connect as a HTTP service and yet have speeds similar to a raw TCP channel.

3.1 Polling issues

The system uses a Wake UP, Sleep algorith to reduce constant polling of the queue. This increases the bandwidth usuage efficiency of the system as unwanted polling is reduced. The wake up and sleep algorithms are as follows:

```
AWAKE_MODE:
1. Request for job.
2. If no job go to SLEEP_MODE.
3. Process the job.
4. Go to step 1.

SLEEP_MODE:
1. Listen to Socket
- If job == AVLB go to AWAKE_MODE.
```

3.2 System Architecture

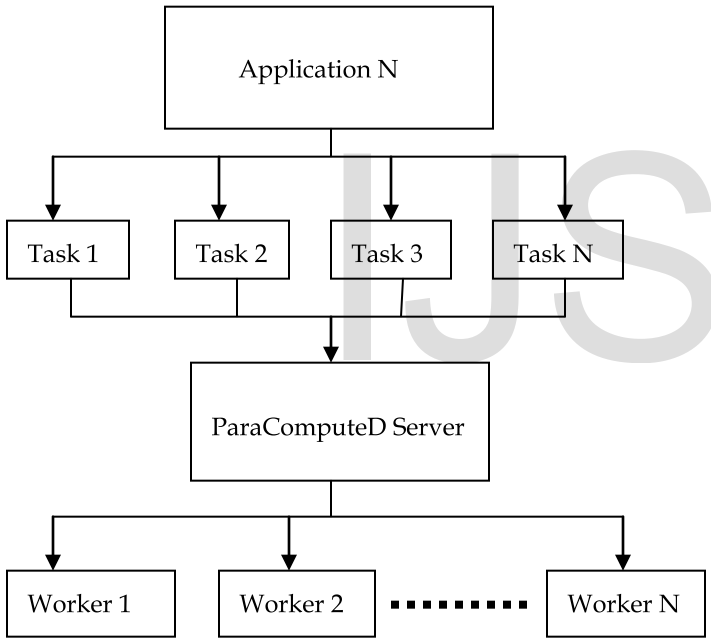

Fig. 3.2.1: System Architecture of ParaComputeD

The above diagram represents the overall system architecture of ParaComputeD. ParaComputeD treats tasks as messages wrapped in Json format. These messages are sent over web sockets which is used as the protocol. When a message is received by ParaComputeD, it decodes it and pushes it inside an internal queue. These queues are named with respect to the workers actions. On the other end, workers are connected to ParaComputeD. These too connect using websocket protocol. The information exchange format is Json. When a worker requests for a job, the ParaComputeD server pops a task from the queue and sends it to the worker that requested it. After the task is complete, the worker sends an acknowledgement back to the ParaComputeD server. THis completes the process for one task. ParaComputeD is built to dispatch and load balance hundreds to thousands of such jobs per minute, making it very efficient at parallel computation. ParaComputeD is also capable of delayed task and also capable of expiring a task after certain interval of time.

For expiring a task, it can stored with the expiry date and time for the task. So the task would get expired after the date and time passes. An apllication can be completed by doin various tasks. Let us take an example of booking a ticket. After the ticket has been booked, let us assume we get an emiled with an attached pdf of the ticket. Thus in this case we have two tasks to be completed. First we have to make a pdf and the send the mail attaching the pdf to it. So the ParaCompue D server would send the two different tasks to two different workers. thus mail would be generated by a different worker and the pdf by a different worker. Thus completing the tasks simultaneously. Also different processes are created for different clients. this way clients need not wait and thus reducing the processing time.

Let us assume clients need to publish or subscribe something from the queue. The for different clients the ParaComputeD would create different processes, so that cdifferent clients can do their tasks simultaneously. Subscribe can be to pop anything from the queue and publish can be used to push anything to the queue.

The algorithms that can be used to publish and subscribe from the queue are:

```
Publish

{
"cmd":"PUB",
"qname":"$Q_NAME",
"message":"$message",
"delay":0,
"expires":0,
"priority":0
}
```

Here, if the command is publish, the message gets pushed to the queue named Q_NAME. We can also set the exipry, delay , and priority for the message.

```
Subscribe

{
"cmd":"SUB",
"qname":"$Q_name"
}
```

This algorithm is used to pop something from the queue named Q_NAME.

The following steps can be included in a queue processing:
1. Client request to subscribe or publish from the queue.
2.Process (P1) gets created for the request made by client in the first step.
3.The process (P1) gets terminated after the event or the request is completed.

4.New processes are created everytime a new event or client request is being made. Two events or client requests can also be made simultaneously. In that case two different processes (P1 and P2) would be made for the two different requests or events.

4 Example

We have seen the typical form submission example in the introduction. That was the case in present processing system. In typical form submission each process is completed one by one in a sequential order. Thus the time required for completion of the whole process just adds up. But the ParaComputeD server proposed by us would reduce the processing time by processing various jobs simultaneously.

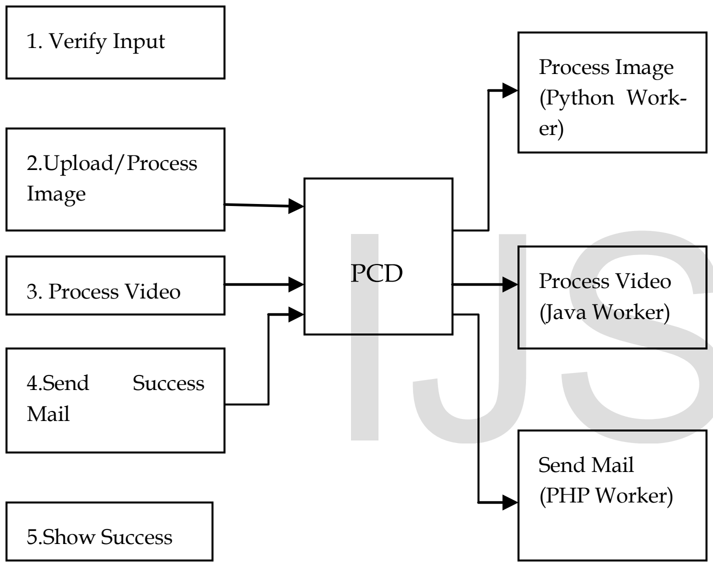

Fig. 4.1: ParaComputeD Form Submission

Here, in case of using ParaComputeD we reduce the processing time by simultaneously processing the image, video and sending mail. Also these processes can be done on different workers. These workers can be running different languages. Hence each process can be given to the worker best suited for the job. Thus, in the above example we have processing image in apython worker, video in a java worker and sending mail in a PHP worker. This ensures best processing as each worker does the job it is best at and the jobs are alotted to the worker best at it.

5 Conclusion

ParaComputeD is a job delegation server which is used farm out work to other machines or processes that are better suited to do the job. It allows you to do work in parallel, to load balance processing and to call function between languages. THere are a number of ways ParaComputeD can be useful. The simplest answer is that you can use ParaComputeD as an interface between a client and a worker written in different languages. You can mix and match any of the supported language interfaces easily; you just need all the applications to be able to understand the workload being sent.

The other way that ParaComputeD can be useful is to put the worker code on a separate machine (or cluster of machines) that is better suited to do the work. Say your PHP web application wants to do image conversion, but this is too much processing to run it on the web server machines. You could instead ship the image off to a separate set of worker machines to do conversion, this way the load does not impact the performance of your web server and other PHP scripts. By doing this, you also get a natural form of load balancing since ParaComputeD only sends new jobs to idle workers. If all the workers running on a given machine are busy, you don't need to worry about new jobs being sent there. This makes scale-out with multi-core servers quite simple: do you have 16 crores on a worker machine? Start up 16 Instances of your worker. It is also seemless to add new machines to expand your worker pool, just boot them up, install the worker code, and have them connect to the existing ParaComputeD.

You are able to run multiple job servers and have the clients and workers connect to a single ParaComputeD.

By using this framework you can scale out your clients and workers as needed. The Daemon can easily handle hundreds of clients and workers connected at once. you can draw your own physical (or virtual) machine lines where capacity allows, potentially distributing load to any number of machines.

Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Authors are strongly encouraged not to call out multiple figures or tables in the conclusion—these should be referenced in the body of the paper.

References

[1] Felician ALECU, "Queuing Systems and Parallel Processing", Economy Informatics, IEEE Transactions on 2003
[2] Parallel Computing: https://en.wikipedia.org/ wiki/Parallel_computing
[3] Erlang: http://www.erlang.org/
[4] Erlang: https://en.wikipedia.org/wiki/Erlang_(programming_language)
[5] Jonathan Weinberg, "Job Scheduling on Parallel Systems", University of Califrnia, San Diego.