

Big Kahuna - A Framework for performing massively distributed computation using grid of stateless transient nodes.

Ritesh M Nayak
International Institute of Information
Technology
26/C, Electronic City, Hosur Road
Bangalore 560100,India
ritesh.m@iiitb.ac.in

Ashwin Rajeev
International Institute of Information
Technology
26/C, Electronic City, Hosur Road
Bangalore 560100,India
ashwin@iiitb.ac.in

GNS Prasanna
International Institute of Information
Technology
26/C, Electronic City, Hosur Road
Bangalore 560100,India
gnsprasanna@iiitb.ac.in

ABSTRACT

This paper explains how to achieve massively distributed computing using many stateless nodes consisting a mix of web browsers, mobile and thick clients which can perform computation on demand. This architecture solves mutually independent problems that do not require synchronization and inter node communication during computation. The arrangement uses javascript enabled browsers which act as multiple nodes connected to a web server acting as the controller. The system can scale rapidly as more browsers connect to the server and can solve a multitude of problems, especially those for which the communication overhead between partial solutions is low and there is little or no inter node communication. This approach makes the client platform independent and inexpensive to implement. The paper also talks about the advantages and disadvantages of such an approach and also how such a technology is feasible for large scale applications such as data mining, pattern recognition and many more.

Categories and Subject Descriptors

C.2.4 Distributed Systems Client/server Distributed applications
C.1.4 Parallel Architectures Distributed architectures

General Terms

Algorithms, Measurement, Performance, Design, Reliability.

Keywords

Distributed computing, map reduce, browser based computing, javascript, AJAX.

1. INTRODUCTION

Almost every major player in the software business is investing heavily on distributed and cloud computing resources. This exponential growth can be attributed to the surplus of data that has been generated, over the last 5 to 10 years, thanks to user generated content being a new category of data that has taken the internet by storm. Massive computing grids and architectures are being built for the purpose of distributed computing, which is a need when you are trying to linearly look through a billion plus pages of information. Map Reduce category of problem solving was a breakthrough in the way problems, especially those related to the web, were perceived. It showed that large subsets of the problems on the web were actually massively parallel in nature and they could be solved in isolated computing clusters. Big Kahuna is a framework which can perform computations on large datasets using the map reduce paradigm of programming. The computation nodes are made up of web browsers, mobile and other indigenous clients which are connected to a web server. In our setup we run the clients on browser. Javascript routines on the browser perform the computation and data is transmitted using AJAX messages.

1.1 Browsers as a Computing Resource

The internet browser has increasingly become the interface of choice for a broad range of applications ranging from multimedia applications to financial applications. The main impetus for this move is the large user base which is familiar with the interface and knows their way around it. The browser is also a heavily underutilized computing resource, most users do not utilize the browser to its complete potential. Almost every person connected to the internet has to use the browser to do any work online, with software as a service being touted as the next step in desktop computing, the importance of the web browser is greater than ever. Big Kahuna is an attempt to build a framework that can build a massively distributed system over the internet in a short period of time and can use the computational power of the Internet browser.

Early research attempts involved trying to use browser based Applets and Web Start architecture, both built on java, to achieve internet wide distributed computing. The technology is now extinct and also the dependence on the presence of a java runtime makes the solution less effective. Early work in this sphere used technologies that were thought to see huge adoption and also become a standard feature of the browsers. But, almost a decade later, these client side technologies have almost become obsolete and have failed to see large scale adoption. However, one technology, Javascript, has seen tremendous growth in its capabilities and has also become a standard feature of every web browser.

Current day distributed systems that work on web scale data are all thick solutions. Most of these solutions come packaged with a distributed file system so as to distribute data, to be worked on, efficiently. The distributed file system enables each of the nodes to work on data available locally to limit network traffic, in the process improve performance of the system.

In our experiments with Hadoop, a popular open source distributed computing framework by Apache, we found performance improvements for large datasets, but for smaller datasets, the initialization overhead and startup would reduce performance. The other problem was the setting up of cluster with mirrored directory structures, common version of Java and other nitty gritty amidst installation woes. Getting started with such a

framework incurs tremendous costs, both in terms of money and effort. Big Kahuna tries to bring down these costs by using a platform agnostic resource like the browser and also removes the installation overhead.

Big Kahuna's server side architecture is designed to be agnostic to the client's implementation. We have successfully used J2ME based mobile clients in addition to browsers to act as computational nodes. Efforts are also on to port the client to the next best language on browsers Adobe Flash.

2. RELATED WORK

There have been previous attempts to implement a distributed computing system over browsers using Java applets, Fletcher, Malhotra : Network of browsers - A multiprocessor computer[6], Peter Capello et al implemented Javelin[5] which is also a browser based parallel computing system and uses java applets on browsers for computation. JAVM[4] by L.F Lau et al is also a similar approach to build an Internet based parallel computing system. Another browser based distributed computing[3] effort was implemented by F.Boldrin et al. which employs a very similar architecture and was developed in parallel at University of Ferrara, Italy. Big Kahuna differentiates itself by being a more complete and mature implementation. Our architecture avoids coupling the client with any specific technology thus making the framework very flexible. As mentioned above we currently have implemented a browser based solution as well as a mobile based solution. A user participating in computation with our framework is completely agnostic to the fact that his browser or mobile device is being utilized, the user is only notified before the start of a workflow.

3. BACKGROUND

In this section we explain the background behind all the technologies we are using to build Big Kahuna. We try to justify our decision in choosing the appropriate technologies.

3.1 Javascript

Javascript is a lightweight, object-oriented, cross-platform scripting language[13]. Javascript, which was introduced primarily for validation of controls in HTML, has seen tremendous development in terms of features. A modern browser can easily access at few megabytes of cache and has reasonable computational power. Frameworks that have been built on javascript have features like exception handling and pseudo multi-threading. Many web based applications mandate use of javascript and a recent statistic shows that almost 85% of the browsers, across the world, have scripting enabled[14]. JavaScript, despite its syntactical similarities with C and Java, has more in common with functional languages like Lisp or Scheme. It implements both procedural and object oriented programming paradigms. Objects are created programmatically in JavaScript, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Javascript runs on a run time environment and these run times are usually implemented within web browsers. There are a number of javascript implementations available, for example, Rhino(Java implementation) and it can be used outside the browser by embedding the engine in an application. If Big Kahuna were to ever move out of the browser then such frameworks would provide interesting alternatives. The

browser thus represents a highly available resource with a few million connected to the web at any given time. The javascript runtime implemented within the browser makes for a very effective tool which can be used to perform computation.

3.2 Asynchronous JavaScript and XML (AJAX)

AJAX is not a technology in itself, but is a term that describes a "new" approach to using a number of existing technologies together, including: HTML, XHTML, javascript, CSS, XSLT, and XMLHttpRequest Object[13]. These technologies help make incremental updates to the content on a browser without reloading a page. The XMLHttpRequest makes background requests to URI's which return either XML or fully formed HTML. This data is then used to render changes to the visual interface on the browser window. AJAX is a very powerful concept and Big Kahuna uses AJAX to ferry control information, data and results across from browser to server and vice versa.

AJAX request reply cycles have been designed to be Asynchronous in nature. The plumbing that AJAX works on is http, and all fault tolerance mechanisms of http are built into AJAX. Since Big Kahuna works on a distributed problem, asynchrony poses a very big problem. The map reduce paradigm works well because there is no need for a consensus in the midst of a workflow, as there is practically no internode communication. The only consensus that would be required would be completion of the workflow which can be decided by the server allocating jobs. Hence problems of asynchrony in traditional distributed systems don't affect Big Kahuna. Control information and Data is sent and received through these AJAX messages. Control information is required so as to decide when and what to compute. Data is transmitted from the server to the client as JSON. Details of JSON are explained in the following section.

3.3 JSON Data

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is built on two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

Big Kahuna uses data in JSON format for transmission between the server and the client. Though AJAX was traditionally built for XML data, when dealing with large amounts of data, the time for marshalling and unmarshalling XML data would be large. JSON data in comparison requires no unmarshalling or parsing. Data in JSON format, if well formed, can be assigned directly to an array and loaded into memory for execution. Though there are very few error correction mechanisms with JSON data, the improvements in terms of speed of execution shadow the viability of XML as a candidate. A sample JSON list is shown below

"Alabama", "California", "Texas", "Delaware"

This can be assigned to a javascript array using a single call to the *eval()* function after which the data can be accessed as an array in javascript. No other data format can achieve this in javascript

which justifies why Big Kahuna uses this for transmitting data to the client browsers.

4. MAP REDUCE

Map Reduce is a programming model and an associated implementation for processing and generating large data sets. It uses Lisp like functions called map and reduce that work on large datasets and many real world tasks are expressible in this model. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines[1]. Simple map routines are run independently on partitioned datasets, the results of which are then collated together in the reduce routines. A simple example of a map reduce routine is given below

```
map(lambda x: x*x*x, range(1, 11))
    [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000
```

```
reduce(lambda x, y: x+y, range(1, 11))
    55
```

Map Reduce differs from standard divide and conquer problems in ways like (1) results of map computations are not required by other computations, so they are mutually independent computations (2) Just like the branching factor in divide and conquer, Map Reduce problems can define a reduce factor. If enough number (reduce factor) of candidates are present to perform a reduce, then the reduce computation can take place along with map computations. Reduces don't have to wait for all the maps to complete before starting.

4.1 Map Reduce in Big Kahuna

Big Kahuna follows the map reduce paradigm for its computation. Both the map and reduce routines are written in javascript which is executed on the client browsers. The web server takes care of scheduling the map and reduce operations and also the data that the routine has to work on. The server sends the data to the client browser which calls the map routine and passes it the data. The result from the map routine is then returned back to the server. Reduce operations also work this way. All the user has to do is write the map and reduce routines in javascript and embed those scripts onto a HTML page for this mechanism to work.

Since browser based http communication is asynchronous in nature, it's almost impossible to predict the liveness or the availability of the client. By Following the map reduce paradigm, we can make the process robust to constraints imposed by the programming model. If a map operation or a reduce operation fails to return a result, it can be requeued for execution by another client. Also, since there is hardly any interdependence amongst individual computations, failure means only a single dataset has to be reworked on.

The class of problems that Big Kahuna is perfectly suited for are those that require little or no internode communication and where the dataset can be split into granular bits big enough to be transmitted though HTTP. The reason for having minimal or no communication between the computing nodes shall become clearer as we elaborate on the distributed environment.

5. BIG KAHUNA - ARCHITECTURE

The design of Big Kahuna had to take into consideration a lot of factors. To achieve true platform and hardware independence the programming model should be decoupled from the complexities of the execution environment by providing a virtual machine abstraction with uniform and predictable semantics. The execution environment needs to implement the virtual machine or work on top of one[2].

The idea was to build a solution that would make the experience seamless to the users. The decisions on technology have already been discussed above. The server side technology chosen was J2EE, mainly because of its wide spread adoption and availability on multiple platforms. Each problem, which is a set of map - reduce computations, is organized as a workflow. The Big Kahuna server side controls all aspects of a workflow.

A typical workflow is started after there are enough clients connected to the server. The clients keep polling the server prior to the workflow signaling that the client is alive and ready to perform computation. Once the workflow administrator starts the workflow, clients are signaled about the starting of the workflow. The clients then proceed to request for data, perform the necessary computation and return the result back to the server. The routine to be called (whether it's a map operation or a reduce operation) is communicated to the client before it requests for data. A typical scenario of a client working is shown below.

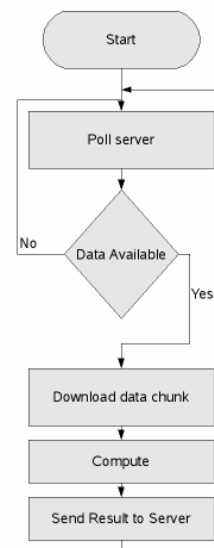


Figure 1. A typical client scenario

5.1 Client

The primary design goal was to develop an OS independent client; We have already mentioned the merits of using a web browser for this purpose. Initially the browser was the client of choice, but as our server side architecture matured, we realized we could accommodate more transient nodes to perform computations. The entire process was achieved by making the calls to the server RESTful. This means that clients need to call specific URI's to get control information and data and these calls

do not need any kind of authentication or special procedures. Any language capable of making HTTP calls can contact the job allocation server and receive control information and data for processing. We have ported the client to java running on a J2ME based mobile phone. Plans are on to extend it to Flash and other languages capable of performing computations. All these clients can then work in synchrony to solve a problem.

To explain a client's working, we shall consider a browser. The same set of steps correspond to non browser implementations. When the user points his browser to the specified URL, he/she is served a web page with pre-loaded javascript code. The client then continually polls the Resource Manager module on the server notifying the server that it is ready to perform computation. When the workflow administrator starts the workflow, the Resource Manager signals the client to request the Core Module for work also indicating whether to execute a MAP routine or a REDUCE routine. In case of a MAP operation, the client then requests the core module for work and a chunk of data is downloaded by the client and the data is passed to the pre-loaded MAP routine for computation. The solution is sent back to the server for aggregation and the client goes back to polling the server. The results from this MAP operation is stored in memory; if there are enough results to perform a REDUCE operation (the reduce factor is decided by the workflow administrator), the set of results are sent to a client requesting for work by indicating that it's a reduce routine. The reduce results are put back into the queue to be aggregated for another REDUCE operation. In this way all the map and reduce operations are completed.

5.2 Server

The Big Kahuna server consists of the modules as described below.

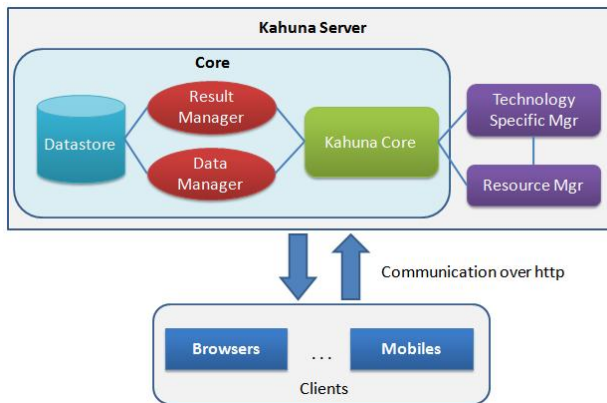


Figure 2. High level architecture

Resource Manager : This module maintains the clients whilst the workflow isn't running. When a client joins the framework, it pings the Resource Manager, which talks to the Technology Specific Manager and allocates it a unique identification number. The client then keeps pinging the Resource Manager indicating that it's alive and is ready to perform computations. Once the workflow administrator starts the workflow, the Resource Manager gives the list of free clients to the Core module and also

informs all the clients to contact the Core Module to get the data for computation.

Technology Specific Manager : Since heterogeneous clients can work together and each have their own requirement, the Technology Specific Manager dictates, through the Resource Manager, what URI's should the client request. This is different for different types of clients. For example, a javascript based browser client may get a URI to request for data in JSON format, while a J2ME client may get URI's for XML or some other sort of data representation.

Data Manager : This module contains all the data files that have to be processed using the Map operations. The Data Manager reads data files and maintains relevant data structures for queuing Map jobs. The data can be present in the relative path of the web application or can reside elsewhere. The workflow administrator can define where the data files reside. If the file is too large to be sent via HTTP, the Data Manager handles splitting of the files into smaller bits. The choice whether to split the files or not, if so, then the granularity of it etc is decided by the workflow administrator. The results from the computations of these map jobs are sent to the Result Manager for further computation.

Result Manager : This module is responsible for collating all the results from the Map operations and queuing them up to be sent as data for the Reduce operation. The reduce factor is decided by workflow administrator and the job of the Result Manager is to concatenate reduce factor number of items in the result queue and send them as data. Once the results are returned, these items are removed from the result queue and the new result is added back to the queue. The workflow is considered complete when all the data queues and processing queues are empty and there is just one item in the Result Queue.

Datastore : The datastore is either an in memory store or a database and is used by both the Data and the Result manager to maintain their corresponding queues. The datastore has an in memory as well as a database implementation. For large queues of large files, the database can be used but typically the performance of such a store on disk makes the processing slower. Efficient caching strategies are used to reduce the number of reads from the database when disk based datastore is used.

Core : The heart of Big Kahuna is the core. The core module contains routines for Job Allocation and scheduling, hand off procedures and also algorithms for fault tolerance. The URI interface to the Core takes care of all these requests and operations and the back end boasts of a robust algorithm to prevent crashes and other mishaps.

The Resource Manager redirects the client to request the Core, the core based on the request consults both the Data Manager and also the Reduce Manager and decides whether to allocate the client a Map job or a Reduce job. Once the job is allocated, the core adds to an allocation datastructure both the job allocated and the client it was allocated to (identified by the unique number given by the Resource Manager). When the results are returned,

the entry is removed from the allocation data structure. If a result is not returned and there are no more items in the queue then it's assumed that the client crashed or was terminated. In such a case the data item is sent to some other client that is requesting for work.

This step is important as transient clients are prone to lot of failure. Failure can occur due to many reasons

1. The data sent to the client may be corrupted in which case the routines will fail to process and return a result
2. The user may close the browser or any other client, thereby killing the computational process.

By ensuring that incomplete jobs are reallocated to different clients the core adds fault tolerance methods to the entire workflow. The web front end presented to the workflow administrator shows the status of the operations at any given time during the execution of the workflow. The administrator can also see a list of clients connected to system and set values for variables like the Reduce factor, data files path, splitting criteria and datastore type.

5.3 Other Issues

Here we address some of the other issues that were considered when designing Big Kahuna.

Security: Since javascript is preloaded into the browser, the code is freely available to the users to see and this may be perceived as a security risk. There are a lot of hacks to get around this problem. One solution is delaying the loading of the javascript code till the workflow starts, the code itself gets transmitted as an asynchronous call and it can be loaded directly into memory and readied for execution. Another solution could be to obfuscate the javascript code that is loaded into the browser. This has two benefits, one is that the code is hidden and if the obfuscation is optimized, the size of the code can be reduced dramatically, thereby saving some bandwidth. Another solution can be to encrypt the data that is being sent to the client.

Liveness: The jobs are allocated based purely on first come first serve basis. Clients that have either crashed or have been terminated won't be allocated any work. The jobs allocated to those clients will be reallocated to other clients. Also, since the overhead of joining a workflow is very low, the number of clients that can join the process can be very large. Given that the map and reduce routines are error free, we can be assured the system will be highly available.

Distribution: Since allocation is first come first serve, jobs are allocated equally to all clients. The faster clients, as a consequence, will get more jobs than slower clients, but that too can be controlled by the workflow administrator.

Scaling out: The network and scalability of the web server will be primary bottlenecks to the performance of the framework. Most of these problems have been worked on for quite some time now and some standard solutions do exist. If the web server proves to be a bottleneck, then you can employ multiple servers to farm out jobs to the clients.

6. PERFORMANCE OF BIG KAHUNA

The following sections discuss the feasibility and performance of our implementation. We also try and explain the kind of problems

for which such an approach is feasible. That was primarily the reason we chose the map reduce paradigm of programs as a suitable candidate. The problem with map reduce is that though the model enforces rules such as no internode communication, no order of execution and division of the input data set, some specific rules for Big Kahuna have to be stated explicitly. Big Kahuna works on http and deals with stateless clients because of which there are constraints. Some of them are :

1. Since the clients are stateless there are additional overheads to establish and ascertain authenticity of a client. The Big Kahuna solves this problem by issuing the client a unique identifier the first time the client connects, and that identifier is used as a session variable for further communication.

2. The method of data transmission in case of a browser is asynchronous which means overhead in sequentializing the process. AJAX requests, though non-blocking in nature, have to be made blocking calls to maintain the sequence of operations.

3. Medium of transmission being HTTP, there are limitations to the number of bytes of information that can be received and sent. Though messages of any length can be received, message length for sending a result is small.

4. Also, type and bandwidth of the network prove to be bottlenecks when it comes to performance. Typical Local Area Networks run 10/100 Mbps connections. The bandwidth keeps getting shared as more and more people join the network and also transmission speeds dip as a result of lot of systems contending to get a hold of the channel.

5. Traditional distributed computing solutions, which are geared for performance, come with a local data store or better yet, a distributed file system. This is required as the amount of data that is required for processing is huge and disk IO becomes a bottleneck when there is a single datastore. Big Kahuna also faces this problem as there is a central data store from which many clients have to be served the data. We try to optimize this by prefetching the file and keeping it in memory before it is requested in order to save on disk IO time. Also, most J2EE servers use efficient caching mechanisms to serve a commonly used page, which also helps when there is so much of HTTP activity.

To get a clear understanding of the type of problems that are best solved using Big Kahuna, we define a metric, the C-L ratio, which we think is a good indicator of whether Big Kahuna is a viable solution to the problem at hand.

C-L Ratio is defined as :

$$C - L \text{ Ratio} = \frac{\text{Computation Time}}{(\text{Latency} + K)}$$

Where,

Computation Time = time to run the core logic of the problem

Latency = Time taken to read the dataset for one Job on disk or time taken to transmit that dataset on the network

k = a constant that accounts for conversions in data formats, parsing delays and other inevitable factors when dealing with textual data.

6.1 3x3 Pattern Search in Inverse Chess logs

One of the candidate problems for running experiments on Big Kahuna was searching for a particular 3x3 pattern in a corpus of logs of the game of Inverse Chess (reference to bib). The game of Inverse Chess was invented and built at IIIT Bangalore [8] and involves playing chess backwards. The game is “playing backwards in moves” rather than the usual chess game. What “playing backwards in moves” signifies is that we start off the game with few pieces on the chess board and continue playing until we arrange all our pieces back to their original positions. An Inverse chess log optimized for the Big Kahuna looks like this

```
1#6, 1#6, 2
6#4, 2#4, 2
```

Where the part before the first # indicates the pawn and the following two sections are initial and final positions of the pawn on the board. Given that the game is played on an 8x8 chess board, our problem involved generating all 3x3 patterns of each state in the game and then looking for the particular 3*3 pattern. As you can see, converting the log to an array representation and then computing 25 of the required 3x3 patterns per move is computationally intensive as the number of moves increase. The data granularity we chose was one log per client. The problem being mainly involving lists and string manipulation, our single CPU comparison was written in python, a language which makes dealing with strings really simple and efficient.

Average Computation Time = 0.3647 seconds
 Average Latency = 0.0002038 seconds
 K approx 0 - very negligible and can be ignored
 Therefore, C-L ratio of Log Analysis = **1795**

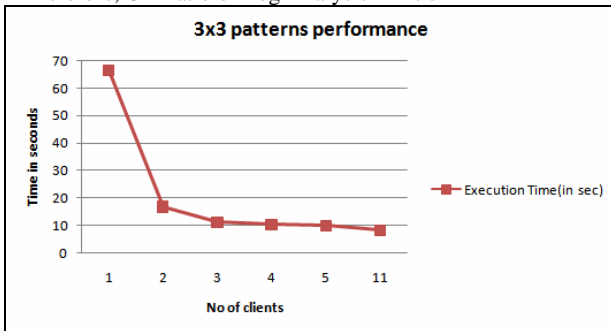


Figure 3. 3x3 speed up using Big Kahuna

6.2 Cosine Similarity of a document in a corpus

Given a document, the problem involves finding the most similar document to the reference document in a large corpus. The problem uses the vector based approach to represent documents and uses the cosine distance between the documents as means of evaluation. Suppose we are comparing two documents, we first vectorize the document based on occurrence of words and then find the cosine of the angle between the two documents. Example :

$$d1 = 1(\text{this}) + 1(\text{is}) + 2(\text{really}) + 1(\text{good})$$

$$d2 = 1(\text{this}) + 1(\text{is}) + 1(\text{bad})$$

Then, cosine of angle between $d1$ and $d2 = \frac{d1 \cdot d2}{|d1| |d2|}$
 where $d1 \cdot d2$ = dot product of the two vectors and $|d1|$ = magnitude of the vector.

For finding the most similar document given a reference document in a corpus of 633 documents, these were the observed values:

Average Computation Time = 0.00353
 Latency = 0.0000590
 Therefore, C-L Ratio = **59.83**

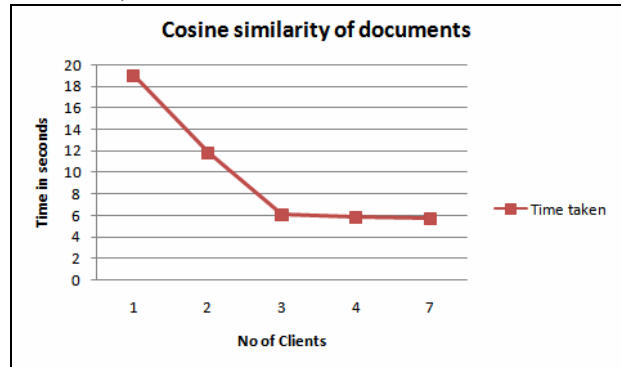


Figure 4. Cosine Similarity speed up using Big Kahuna

6.3 Greatest ten numbers

If finding the greatest 10 of 500,000 numbers were our problem, then we would split the data into sets of 5000 numbers and define our Map and Reduce operations. Map operation would be to find the greatest 10 in a single set of 5000 numbers and the reduce operation would be the exact same thing but on the results of the map operation. Typically 100 results from the map operation are sent as the dataset for the reduce operation. For this problem, the C-T Ratio is calculated on a single processor machine as follows:

Computation time (greatest 10 of 5000) = 0.450918746
 Latency = time to read the file containing 5000 numbers (this since there is no network factor) = 0.5892 seconds
 K = data format conversion time = approx zero

Therefore, C-L Ratio = **0.7652**

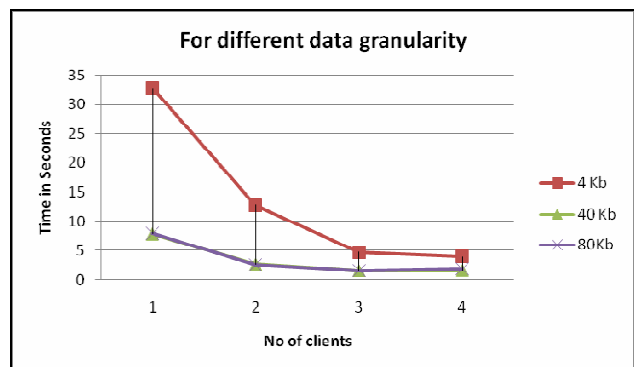


Figure 5. Sorting speed up using Big Kahuna

Given the performance of the problems on the Big Kahuna in correlation with the metric, we can draw some conclusions regarding the problem type and also its feasibility.

If the C-L Ratio is considerably high then it means the computation time dominates the communication time and it's viable to be computed by Big Kahuna

Problems with C-L Ratio > 1 and not very high, the computation time is almost comparable to the communication time and as more and more clients are added, the latency keeps increasing due to the bandwidth being shared amongst the clients. As a consequence, the C-L Ratio keeps plunging and peaks at a value nearing 1. After that, any more clients added will only bring down performance. As more clients are added the value of the C-L Ratio decreases exponentially.

In case of the 3x3 patterns, owing to the high value of ratio, there is significant speedup achieved as compared to a single processor machine and the platform can scale to a large number clients. In case of the cosine similarity, since the ratio wasn't very high, though performance is marginally better compared to a single processor, the framework doesn't scale well and peaks out very fast. In case of sorting, the single processor is way faster compared to the framework.

The size of the dataset also determines the speedup achieved using the framework. As indicated by figure 5, if the dataset is too large then the latency will increase, not only due to the time lag in transmitting but because other clients will use the same channel to receive similar size files. The granularity of the dataset should be chosen optimally. The size of the result should also be small as the result has to be sent back to the server. Large results from Map operations will significant slowdown and such problems should be avoided.

7. CONCLUSION

We have shown how a framework like the Big Kahuna can be used to build a robust and highly scalable distributed computing platform. Contrary to speculation, we have also shown for certain class of problems, this application level network can scale to a large number of clients and also achieve significant speedup. Network bandwidth is a big bottleneck limiting performance but given that gigabit networks are already a reality and faster networks are being designed, we can be assured that this problem will slowly fade away.

No installation, platform independence, ability to scale to a large number of clients and fact that the experience is seamless to a client make this platform very viable for building massively distributed computing systems. The framework also handles issues like managing clients, jobs, faults etc and insulates the user from all these problems. All the user has to do is write the map and reduce routines for a problem and the framework takes care of the rest. Given the current technological bottlenecks, choosing the right problem to be solved still remains an act requiring prudence. We have tried to define a metric and have also listed out pitfalls to help the decision making process easier.

If cloud computing does take off and the world goes thin, install and use softwares will be passé, then a solution like this could harness the power of a large number of clients connected to the system without any of the traditional overheads. There is also a steep increase in the number of mobile clients, tablets and other devices with networking capabilities joining the world wide network, the internet, and a cross platform solution like this can work across all of these ubiquitous platforms to build a highly flexible distributed computing solution.

8. REFERENCES

- [1] Jeffrey Dean and Sanjay Ghemawat - Map reduce - Simplified data processing on large clusters. Communications of the ACM, vol. 51, no. 1 (2008), pp. 107-113
- [2] Arash Baratloo, Mehmet Karaul, Zvi Kedem, and Peter Wyckoff. Charlotte: Metacomputing on the web. In Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, 1996.
- [3] F. Boldrin, C.Taddia, G. Mazzini : Distributed Computing Through Web Browser - Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th Volume , Issue , Sept. 30 2007-Oct. 3 2007 Page(s):2020 - 2024
- [4] LF Lau, AL Ananda, G Tan, WF Wong, "JAVM: Internet-based Parallel Computing Using Java", Series on Scalable Computing - Vol 2, Annual Review of Scalable Computing, Singapore University Press/World Scientific, Pg 59-74. December 2000.
- [5] Michael O. Neary, Bernd O. Christiansen, Peter Cappello, and Klaus E. Schauer : Javelin: Parallel Computing on the Internet - Future Generation Computer Systems Volume 15 Issue 5-6 (October 1999) Special issue on metacomputing Pages: 659 - 674 Year of Publication: 1999
- [6] Luke Fletcher and Vishv Malhotra : NETWORK OF BROWSERS – A MULTI-PROCESSOR COMPUTER - Proceedings of the IASTED International Conf. on Parallel and Distributed Computing and Networks February 17-19, 2004, Innsbruck, Austria.
- [7] Atkinson, A (2003) Coalescing Idle Workstations as a Multiprocessor System using JavaSpaces and Java Web Start. Honours thesis, University of Tasmania.
- [8] Soumen Chakrabarti Mining the Web : Discovering knowledge from hypertext data –2005
- [9] The game of Inverse Chess - USPTO Application #: 20080197569
- [10] Homepage of JSON <http://www.json.org>
- [11] Hadoop - HDFS and Map Reduce Solver - <http://hadoop.apache.org/core/>
- [12] Folding@Home - <http://folding.stanford.edu/>
- [13] Mozilla Developer Center
- [14] W3C Counter - www.w3counter.com