Applying Twister to Scientific Applications

Bingjing Zhang[1, 2], Yang Ruan[1, 2], Tak-Lon Wu[1, 2], Judy Qiu[1, 2], Adam Hughes[2], Geoffrey Fox[1, 2]
[1]*School of Informatics and Computing,* [2]*Pervasive Technology Institute*
Indiana University Bloomington
{zhangbj, yangruan, taklwu, xqiu, adalhugh, gcf}@indiana.edu

Abstract

Many scientific applications suffer from the lack of a unified approach to support the management in execution and inefficiency in processing large-scale data. Twister MapReduce Framework, which not only supports traditional MapReduce programming model but also extends it with iterations, tries to address these problems. This paper describes how Twister is applied to several kinds of scientific applications such as BLAST, MDS Interpolation and GTM Interpolation in non-iterative style and MDS without interpolation in iterative style. The results show the applicability of Twister to data parallel and EM algorithms with small overhead and increased efficiency.

Keywords

Twister, Iterative MapReduce, Cloud, Scientific Applications

1. Introduction

Scientific applications are required to process large amount of data. The volumes of Input data grow from gigabytes to terabytes, even petabytes scale now. This already far exceeds the computing capability of one computer. Although the computing tasks can be parallelized on several computers, the execution may still take days or weeks long.

This situation demands better parallel algorithms and the distributed computing technologies which can manage the scientific applications efficiently. MapReduce Framework [1] is a kind of technology which becomes popular in recent years. KeyValue pairs make the input be distributed and parallel processed at a fine granularity. The combination of Map tasks and Reduce tasks satisfies the task flow of most kind of applications. And these tasks are also well managed under the runtime platform.

This paper introduces Twister MapReduce Framework [2], an expansion of traditional MapReduce Framework. The main characteristic of it is that it does not only support non-iterative MapReduce applications but also iterative MapReduce programming model efficiently to support Expectation-maximization (EM) algorithms with communication complications, which is common in scientific applications but is not allowed by other former MapReduce implementations such as Hadoop [3].

Twister uses publish/subscribe messaging middleware system for command communication and data transfers. It supports MapReduce in manner of "configure once, and run many time" [2]. Data can be easily scattered from client node to compute nodes and combined back into client node by APIs. With these features, Twister can support iterative MapReduce computations efficiently when compared to other MapReduce runtimes. Twister is also compatible with Cloud architecture. Now it has been successfully deployed on Amazon EC2 platform [4].

In this paper, the applicability of Twister is mainly discussed. Through implementation of several scientific applications, this paper shows how these applications are well supported by Twister. In the following sections, the overview of Twister is firstly presented with introducing its programming model and architecture. Then four Twister scientific applications are discussed. Three of them are non-iterative programs which are Twister BLAST, Twister GTM Interpolation, and Twister MDS Interpolation. The final one is Twister MDS which is an iterative application. Workflow and parallel mechanism supported by Twister are presented within this section. The conclusion is drawn in the final section.

2. Twister Overview

This section gives an overview to Twister MapReduce Framework. The first part illustrates how non-iterative and iterative MapReduce programming model are supported in Twister. The second part describes the architecture of Twister.

2.1. Non-Iterative and Iterative MapReduce Support

Many parallel applications are only required to do Map and Reduce once, such as WordCount [1]. However, some other applications are inevitable to be in an iterative pattern such as Kmeans [5] and PageRank [6]. Their parallel algorithms require the program to do Map and Reduce in iterations in order to get the final result.

The basic idea of Twister is to let MapReduce jobs only be configured once, then let it run in one turn or

several turns according to the client's request. If there is only one turn execution, it is exactly the same as non-iterative MapReduce. The result is produced from Reduce method directly. For iterative MapReduce, the output from "Reduce" is collected by "Combine" method at the end of each iteration. A client will send intermediate results back to compute nodes as new input of KeyValue pairs in next iteration of MapReduce tasks (See Figure 1.).

Another important characteristic of many iterative algorithms is that some sets of input data are kept static between iterations. In Twister, these static data are allowed to be configured with partition file, loaded into Map or Reduce tasks, and then being reused through iterations. This mechanism significantly improves the performance of Twister on iterative MapReduce computing and makes it different from those methods, which mimic iterative MapReduce by simply re-executing MapReduce tasks without caching and reusing data and job configuration. In addition, because the data cached inside of Map and Reduce tasks is static, Twister still keeps "side-effect-free" nature [2].

In this workflow, Twister also provides fault tolerance solution for iterative MapReduce programming model. Twister can save the execution state between iterations. If Twister detects faults in execution, it can be rolled back few iterations and restart computing.

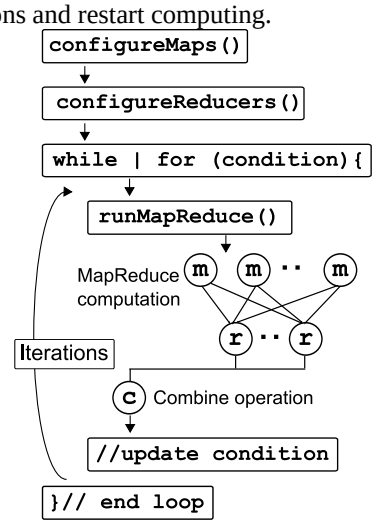

Figure 1. Twister MapReduce Workflow [2]

2.2. Architecture

Twister has several components. Client side is to drive MapReduce jobs. Daemons and workers which live on compute nodes manage MapReduce tasks. Connection between components are based on SSH and messaging software.

To drive MapReduce jobs, firstly client needs to configure the job. It configures MapReduce methods to the job, prepares KeyValue pairs and configures static data to MapReduce tasks through partition file if required. Once the job is configured, client can run the MapReduce job and monitor it to completion. Between the iterations, it receives the results collected by Combine method. When the job is done, it terminates the job.

Messages including control messages and KeyValue pair data are transmitted through a network of message brokers with publish/subscribe mechanism. With a set of predefined interfaces, Twister can be assembled with different messaging software. Currently Twister supports two kinds of software. One is NaradaBrokering [7], another is ActiveMQ [8].

Daemons operate on compute nodes. They load Map Class and Reduce Class and start them as Map and Reduce workers which can be also called Mapper and Reducer. In initialization, Map and Reduce workers load static data from local disk according to records in partition file and cache the data into memory, and then they execute Map or Reduce function defined by users. Twister uses static scheduling for workers in order to let the local data cache be beneficial to computing [2]. So this is a hybrid computing model. Inside one node, workers are threads and managed by daemon processes. Between nodes, daemons communicate with the client through messages.

Twister uses scripts to operate static input data and some output data on local disks in order to simulate some characteristics of distributed file systems. In scripts, Twister uses "scp" command to distribute static data to compute nodes and create partition file by invoking Java classes. For data which are output to the local disks, Twister uses scripts to collect data from all compute nodes to a node specified by the user.

3. Twister Non-Iterative Applications

Twister can support non-iterative applications which are in style of "Map and then Reduce" or "Map only". "Map and then Reduce" is a normal case in traditional MapReduce programming model. A classical scenario to use this model is WordCount. Every Map task calculates the word count in local partial text, and sends the intermediate results to Reduce tasks with the word as Key, the count as Value. Then Reduce tasks collect the partial result and get the total count of one word. Meanwhile, "Map only" means data are processed by parallel Map tasks and then output directly. This parallelism method is also frequently used.

In "Map only" style applications, a common way to do computing parallel is to invoke the execution binary of a stand-alone version program, usually called binary invoking mode. This method is often used in parallel applications for several reasons. Nowadays many stand-alone scientific programs are complex and updated frequently with new features. In this situation, rewriting the parallel version of original stand-alone program may

take much effort and cannot catch up with stand-alone version in new features. Due to these reasons, binary invoking becomes a considerable solution. MapReduce framework makes this solution applicable because it can well handle input data split and manage the parallel task execution.

Here three new non-iterative MapReduce applications, including Twister BLAST, Twister MDS Interpolation, and Twister GTM Interpolation, are introduced in the following sections.

3.1. Twister BLAST

Twister BLAST is a parallel BLAST application based on Twister MapReduce framework. Here this section will introduce the nature of BLAST software, related other parallel BLAST applications, and the characteristics of Twister BLAST. Finally, a performance comparison is given between Twister BLAST and Hadoop BLAST with detailed analysis.

3.1.1. BLAST Software

BLAST [9] is a stand-alone local gene search tool. It has two versions. One is BLAST which is written in C. Another is BLAST+ which is written in C++. BLAST+ is the latest version of BLAST and is recommended by NCBI. Because of this, the term BLAST used below mainly points to BLAST+. The version used here is 2.2.23.

BLAST is a command line tool which accepts input parameters and output the result to screen or file after its execution. There are two important inputs [10]. One is query location, another is database location. BLAST query is a file which contains FASTA-format gene sequences which will be searched through the whole database. BLAST database is a set of formatted files which contain gene data and organized with indices. The total size of the database is usually large, which can be gigabytes level. Once BLAST receives database path and query path, it will do BLAST search. BLAST search has three phases [11]. The first phase is “Setup”. The query is read into the memory and a “lookup” table is built. The next phase is “Scanning”, each subject sequence in database is scanned for the words matching the query in “lookup” table. The final phase is “Trace-back”. Improved score and insertions/deletions are calculated for query sequences.

BLAST is a system resources demanding application. On a IU PolarGrid [12] node with two 4-core CPUs (Intel Xeon CPU E5410 2.33GHz) and 16 GB memory, searching hundreds of gene sequences with 37 gene letters each through a 10 GB NR database [13], BLAST consumes one core’s 100% CPU and 20% memory of the total under one-thread mode. It can exhaust all memory on a machine if the input is too large or if there are too many hits to the database [10].

BLAST can also be executed under multi-thread mode. Under this mode, it can utilize multi-core but still uses 20% memory. However, it won’t utilize eight cores fully at all the time. For example, on the node with settings mentioned above, executing BLAST with 8 threads, CPU usage is not always 800% but occasionally dragged down. The reasons is that BLAST is only multi-threaded in its “Scanning” stage. The chart below shows the execution time comparison and the speedup of using 8-thread mode under different input size. The speedup value is greatly affected by database loading time when the input size is small and then become stable as the input size growing larger than 100 sequences. However, all the values are below than 7.8, which is still lower than 8. This means using multi-thread mode will not be as efficient as multi-process mode in the case that the node can provide enough memory for multi BLAST processes execution (See Figure 2.).

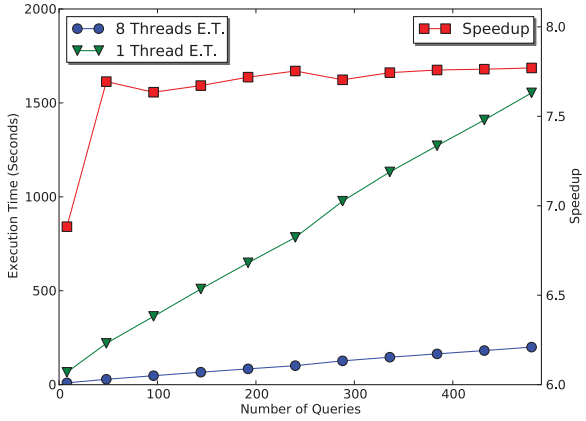

Figure 2. Execution Time and Speedup between 1 Thread and 8 Threads on One Node under Various Input Size

3.1.2. BLAST Parallelism Method

Several kinds of parallel BLAST applications are already implemented, including MPI BLAST [14], Cloud BLAST [15], and Azure BLAST [16]. This section will introduce these technologies through timeline.

MPI BLAST uses MPI library [17] to support parallel BLAST. It modifies original BLAST program through combining NCBI toolkit [18] and MPI library together. Query and database are both partitioned. Once MPI BLAST starts, it distributes database partitions to all compute nodes, and then uses one process to dynamically schedule query chunks to different workers. Because of database segmentation, every worker cannot produce a complete output. As a result, one process is use to merge the result and output to a shared directory. The reason why database is segmented is that MPI BLAST designers

believe that the database is too large so that they cannot be put into memory or even hold on local disk [19]. However, database segmentation also generates lots of communication work and nowadays modern clusters have large memory and disks which can easily hold 10-gigabytes level database volumes. Besides, the latest version only supports an old BLAST version which lacks new features and slow in performance. Recent experiment which has the same data and node settings as above shows that, by using MPI BLAST 1.6.0 Beta1, 100 gene sequences on one node with 10 MPI processes (8 workers + 1 scheduler + 1 merger) consumes near 400 seconds while using the latest BLAST+2.2.23 only needs about 60 seconds in 8-thread execution mode. Furthermore, MPI BLAST doesn't have fault tolerance support, which is fatal because BLAST jobs usually require long execution time. Based on these reasons, it can be concluded that MPI BLAST is outdated and obsolete.

Cloud BLAST uses Hadoop MapReduce Framework to support parallel BLAST on cloud platform. Hadoop is used here for resolving issues like data splitting, worker finding, load balancing, and fault tolerance [15]. MapReduce computing is used in "Map only" style. Original data are split into small chunks and distributed to workers. On each node, input data chunk are processed by invoking BLAST binary and searching through a local database copy. The outputs are stored in HDFS [20]. With this computing style, Cloud BLAST has less cost in communication. It has been proved that this kind of architecture has better performance than MPI BLAST, and is even scalable, maintainable and fault tolerable [15].

Azure BLAST is nearly the same as Cloud BLAST in computing style. But it is directly supported by Azure Cloud Platform [21] rather than a MapReduce framework. However, compared with Hadoop, Azure platform still provides similar functionalities such as data splitting, worker finding, load balancing, and fault tolerance.

3.1.3. Twister BLAST Solution

Twister BLAST is a parallel BLAST application which is supported by Twister. Based on the analysis about three parallel BLAST applications above, Twister BLAST also uses binary invoking parallelism in order to keep Twister BLAST in state of art. As what already analyzed, this style brings scalability and simplicity to program and database maintenance. The flexibility of Twister framework allows this program to run on a single machine, a cluster, or Amazon EC2 cloud platform.

Before Twister BLAST execution, query chunks are distributed to all compute nodes through Twister scripts because the gene query could be large in size and not be able to be loaded into client's memory together and then be sent as KeyValue pairs. Later on, a partition file will be created to record the location of these query chunks. It will replace KeyValue pairs and be configured to Map tasks as input information.

BLAST Database is also replicated to all the compute nodes. Though moving entire database among network may cost much, however, it is easy to manage database versions and brings efficiency for later BLAST execution. In order to replicate the database through network quickly, compression techniques is used here. BLAST Database, such as 10 GB NR database, will be compressed into 3 GB and then be distributed. Once they arrive at compute nodes, they will be parallel extracted through a set of Map tasks. This will significantly reduce the time needed by replication to one third of the original time.

Twister BLAST also uses Map tasks to parallelize BLAST jobs. Twister BLAST client sends job property messages through a set of message brokers to drive Map tasks. Then Twister will start Map tasks according to the partition file. Each Map task will invoke BLAST program with query file location and other input command variables defined by user. Once jobs are done, Twister will report the status to the client program. Outputs can be collected to one node by Twister scripts (See Figure 3.).

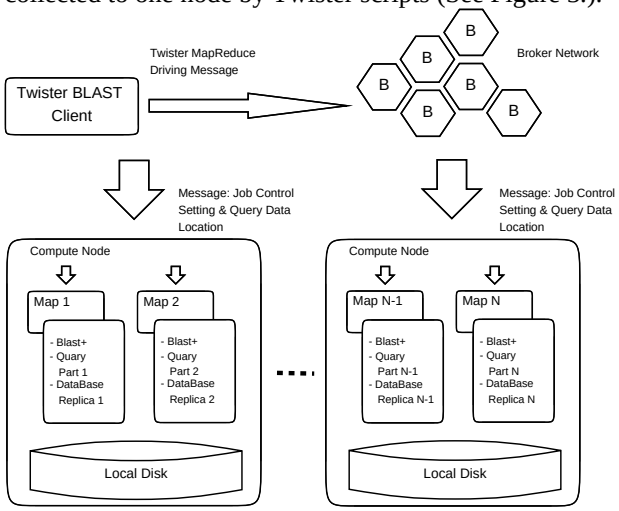

Figure 3. Twister-BLAST Workflow and Architecture

In addition, another important fact observed by domain experts may give us a chance to extend Twister BLAST solution. Gene queries generated by Bioinformatics researchers can easily contain duplicates. There are already several tools to remove the duplication [22-26]. However, there is no scalable solution to handle large inputs. Here Twister can be used to solve this problem by using a WordCount like MapReduce job before doing parallel BLAST job. Once the original query are partitioned and distributed to all nodes, Map tasks can remove local duplicates, then send KeyValue pairs, with each of which uses a gene sequence as Key and a tag as Value. After receiving these KeyValue pairs, Reduce tasks can generate non-duplicate gene sequences with a unique tag. If assuming this result can be much less than

the original data size, we can use Twister Combine method to collect these gene sequences back to the client and then re-assign them to KeyValue pairs and send them back to Map tasks to do parallel BLAST. Depending on the quality of the inputs, Twister BLAST can probably save quite amount of time.

3.1.4. Performance Tests and Comparison between Hadoop BLAST

A set of performance tests is also done on Indiana University Polar Grid Cluster by using 32 nodes. Each of them has two 4-core CPUs (Intel Xeon CPU E5410 2.33GHz) and 16 GB memory, and a Gigabit Ethernet network interface. Here Twister BLAST is compared with Hadoop BLAST implementation.

Hadoop BLAST basically has the same style as the implementation mentioned in Cloud BLAST paper. It uses HDFS to hold compressed BLAST program and database, and then uses distributed cache to allocate them to local disk. Then it equally splits the query file into sequence chunks, and copies them to HDFS. Once program and data are prepared, they are downloaded, extracted and taken as input by assigned Map task.

Query sequences are selected from the data provided by Center for Genomics and Bioinformatics [27]. It consists of 115 million sequences and each of them has a uniform length of 37 DNA letters. For fairness, removing duplicates are not considered in the experiment. The BLAST job is parallelized by using 256 map tasks. By changing input size, the time trend shows how time grows with the input size.

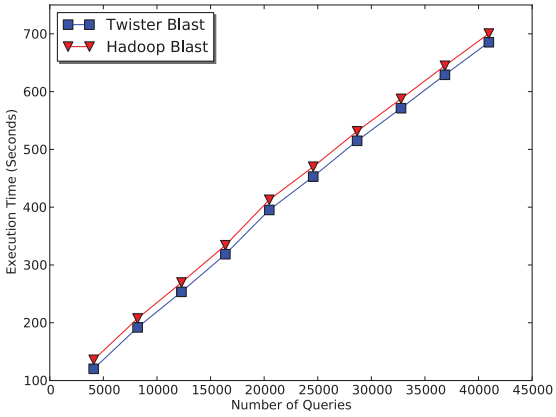

Figure 4. Performance Comparison with Twister BLAST and Hadoop BLAST on 32 PG nodes

For preparing data, in Twister BLAST, NR database replication and query distribution took 1006 seconds for transferring 2.9 GB compressed data and extracting them, while Hadoop BLAST uses 693 seconds. For BLAST execution stage, the result, as drawn in the following figure, shows that the execution time is proportional to the number of gene sequences. Compared with Hadoop BLAST, Twister BLAST has little overhead to computation and is also slightly faster than Hadoop BLAST (See Figure 4.).

However, because of Twister's static scheduler, it cannot dynamically schedule queries to Map tasks. In the experiment, due to the characteristics of queries, the result shows Map tasks have different execution times and the final execution time is decided by the longest Map task execution time. By randomizing the input data, this issue can be eased but not solved.

3.2. Twister MDS Interpolation

Twister MDS Interpolation is a parallel method for MDS Interpolation by using the Twister Framework. We have implemented this program and test it.

3.2.1. MDS Interpolation

The multidimensional scaling (MDS) [28] is known as a dimension reduction algorithm which is widely used in statistics analysis. It is usually used in investigating large data points which may arrive 100k in quantity. However, if this algorithm is computed directly, its time complexity arrives at $O(N^2)$ level, where N is the total number of points. And because holding and calculating matrix needs large memory, this algorithm is also memory-bound. As a result, it is very difficult to run MDS over 1 million data points. Now, with MDS interpolation, the problem of this algorithm can be solved by processing full dataset based on the result from a subset of it.

MDS interpolation is an out-of-sample problem [29]. The subset which has result from MDS is the sample, and the rest of the dataset is out-of-sample points. The time complexity of MDS interpolation is $O(KM)$, where K is the number of sample points and M is the number of out-of-sample points. This greatly reduces the time required to do dimension reduction of MDS and makes processing millions of points be possible.

In order to find a new mapping position for an out-of-sample point, we first do normal MDS on selected n points as sample points from the full dataset to reduce the dimension to L, and then select k nearest neighbors $p_1, \cdots, p_k$ from the sample points for an x from the out-of-sample points. By using this information, we can construct a STRESS function and minimize it. This method which is similar to a MDS algorithm is known as SMACOF [30]. Since only one point is movable among the sample points, we set weight to 1 to do simplification. The STRESS function is

$$\sigma(X) = \sum_{i<j\leq N} \left(d_{ij}(X) - \delta_{ij}\right)^2 = C + \sum_{i=1}^{k} d_{ix} - 2\sum_{i=1}^{k} \delta_{ix} d_{ix}$$

Here δ_{ij} is the original dissimilarity value between p_i and x, d_{ix} is the Euclidean distance in L dimension between p_i and x, and C is a constant.

According to Seung-Hee Bae's method [31], we can minimize this STRESS function by the following equation.

$$x^{[t]} = \overline{p} + \frac{1}{k}\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(x^{[t-1]} - p_i)$$

Here $d_{iz} = \left\|p_i - x^{[t-1]}\right\|$ and $\overline{p}$ is the average of k sample points' mapping results. The stopping criteria for this algorithm would be

$$\Delta\sigma\left(S^{[t]}\right) = \sigma\left(S^{[t-1]}\right) - \sigma\left(S^{[t]}\right) < \theta$$

Here $S = P \cup \{x\}$ and θ is the given threshold value. Then we take this $x^{[t]}$ as our result.

3.2.2. Parallel MDS Interpolation Approach

There are already some types of parallel MDS interpolation methods [32], such as the applications under MPI.net [33] and Dryad [34]. But this time we are going to show how to use Twister to do it. Even though MDS interpolation can dramatically reduce the time required doing the dimension reduction computation, the memory issue cannot be solved by the algorithm itself because 1 million distance matrix file could be up to 6 TB and it is very costly to move this distance file around compute nodes. As a result, in Twister MDS interpolation, an algorithm as vector-based algorithm is implemented, where raw dataset but not Euclidean distance dataset is read. The raw-data file is split into equally sized files and distributed over compute nodes. Twister will use partition file to locate of the raw data chunks. Then Twister MDS Interpolation will create map tasks on each node. Then Twister will use "Map only" mode to start map tasks according to the data locations in the partition file. Data are processed by functions encapsulating MDS interpolation algorithm in each map task and output can be collected by Twister Script.

3.2.3. Performance Test

Performance tests are done for Twister MDS Interpolation on Indiana University PolarGrid which is mentioned in Section 3.1. The numbers of nodes used in tests are 8 nodes, 16 nodes and 32 nodes. Accordingly they are 64 cores, 128 cores and 256 cores. The data is from PubChem [35]. Its original size is 18 million data points. In the experiment, we take 4 million and 8 million data points from it (See Figure 5.).

In Figure 5, parallel efficiency is used as right y-axis and computation time is used as left y-axis. The x-axis is core number. The efficiency of computing is calculated as following:

$$ParallelEfficiency(\eta_i) = \frac{T(p_1)}{\alpha.T(p_i)}$$

Here $T(p_i)$ is the execution time on i nodes, p_1 is the smallest nodes running the program, and $\alpha = p_i/p_1$.

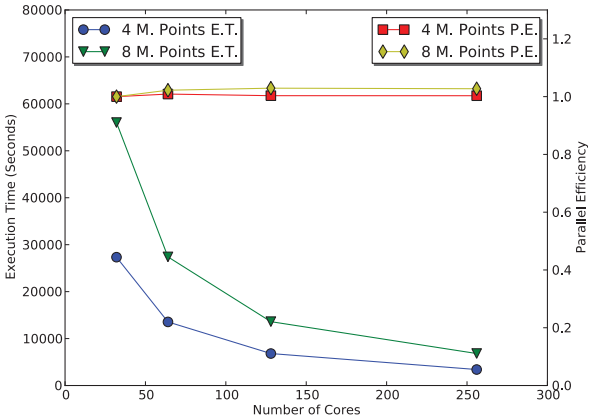

Figure 5. Twister MDS Interpolation Execution Time and Parallel Efficiency

The parallel efficiency is around 1 even when the number of cores is increasing; this is because there is no communication between nodes when we run the MDS Interpolation in parallel. So, with increasing number of cores, Twister MDS interpolation performs better.

3.3. Twister GTM Interpolation

Twister GTM Interpolation is a new application of parallelizing GTM Interpolation. We use the binary GTM program and information from the results of running the GTM to design the new program.

3.3.1. GTM Interpolation

Generative Topographic Mapping (GTM) algorithm is to find an optimal representation of data from high dimensional space to low dimensional space. It seeks a non-linear mapping of user-defined K points in the low dimensional space for N data points in a way that these K points can represent the N data points in the original high dimensional space [36]. So the time complexity of this problem is $O(KN)$. Although this algorithm is faster than MDS since $K < N$, it is still a challenge to compute large dataset, such as 8 million data points.

To solve this issue, GTM Interpolation chose to do normal GTM on a subset of the full dataset, known as samples at first. The remaining out-of-sample data can be learnt from previous samples. Since the out-of-sample data doesn't involve in the computing intensive learning process, GTM Interpolation can be very fast. However, for more complicated data, there are some complexes ways to interpolate GTM [37-39].

According to Jong Choi's work, a simple interpolation approach can be done by the following method. For example, to process 26 million data points, firstly 100k data is sampled from the original dataset. Then GTM is performed on this sample data to find an optimal K cluster center and a coefficient β for this sample set. This information is stored in several files. After that, for the remaining out-of-sample data M , a $K \times M$ pairwise distance matrix D is computed with d_{ij} which is a Gaussian probability between the sample data and out-of-sample data. So the responsibility matrix R can by compute as

$$R = D\emptyset(ee^tD)$$

Here $e = (1, \cdots, 1)^t \epsilon R^k$ and $\emptyset$ represents element wise division.

With this information, finally we can construct a GTM map $\overline{Z} = R^tZ$ as Z is the matrix represents of the sample points.

3.3.2. Parallel GTM Interpolation Approach

GTM Interpolation has also been paralleled by using Dryad, Hadoop and Amazon EC2 [32], this time we are going to use Twister to parallel this program. The Twister GTM Interpolation can divide the raw data file from the out-of-sample data file. And each partition will have a mapper created to process that chunk. Once this is done, Twister will invoke each GTM Interpolation with each chunk to process the data. The mappers will process each block individually, however, we can collect the results by using a different script.

3.3.3. Performance Test

The performance test is also done on Indiana University PolarGrid. 4 million and 8 million data points from the PubChem data [35] are selected, and the sample data size is 100k.

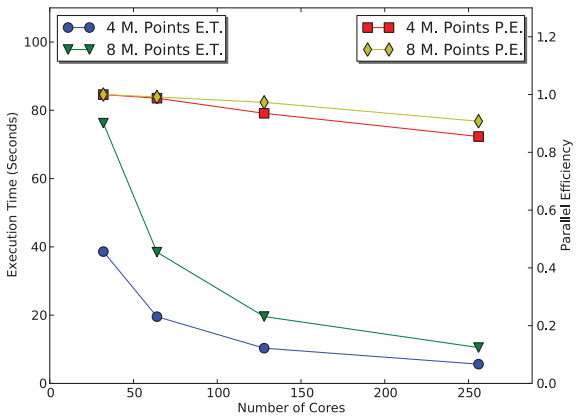

Figure 6. Twister GTM Interpolation Execution Time and Parallel Efficiency

As can be seen in figure 6, GTM-Interpolation run very fast on PolarGrid, it takes 76 seconds to run on 4 nodes, 32 cores, and twister's parallel efficiency remains above 0.85, which is fairly high for parallel program. And we anticipate that with increasing number of cores, even above 256 cores, the parallel efficiency will remain above 0.8 and become more stable.

4. Twister Iterative Applications

The unique feature of Twister is to support iterative MapReduce programming model. Client can drive Twister to finish MapReduce job in iterations. The performance is optimized by caching static data in computation and using message infrastructure in communication. Faults are handled between iterations. Here Twister MDS is introduced to illustrate how iterative MapReduce works in Twister.

4.1. Twister MDS

Multidimensional scaling (MDS) is a set of algorithms which can map high dimensional data to low dimensional data with respect to the pairwise proximity information. In this algorithm, the pairwise Euclidean distance within the target dimension of each pair is approximated to the corresponding original proximity value. This procedure is called STRESS [40]. It is a non-linear optimization algorithm to find low-dimensional dataset which minimizes the objective function.

Because a large high dimension distance matrix is involved in, MDS is mainly a kind of data intensive computing. The following part will show how iterative MapReduce programming model can be applied to this algorithm to facilitate its execution. Here Twister MDS application is implemented and its performance and scalability is evaluated.

To reduce the memory requirement on single node, the original distance matrix is partitioned into chunks by rows. These chunks are distributed to all compute nodes, and the partition information is recorded in a partition file. These data chunks are assigned to Map tasks in one to one mapping. Once they are configured to Map tasks, they will be held in memory and used through iterations.

Twister MDS shows how the concept "configure once and run several times" works. After initialization, it configures three jobs to Twister. Two of them are matrix-vector multiplications and the other is STRESS value calculation. Once these jobs are configured, client begins to do iterations. In each loop, it will invoke these three jobs sequentially. The matrix result obtained from the previous job is collected by the client and used as KeyValue pairs input in the following job. Since the intermediate matrix result is required by all Map tasks of the next job according to the algorithm, they are sent

through **runMapReduceBCast** method which can broadcast the data value to all nodes with different keys. Once a loop is done, the mapping matrix result and STRESS values are used as input for next loop. Client can control the number of iterations. Once the max iteration arrives, the client stops computing.

To evaluate performance of Twister MDS, a Twister environment with one ActiveMQ message broker established. Twister MDS runs with 100 iterations. A metagenomics dataset comprising of 30000 data points with near 1 billion pair-wise distances is tested here. Because the data of this large size cannot be handled on single machine, the method for calculating parallel efficiency used in sections above is applied again; this means parallel efficiency is calculated with respect to the minimum number of CPU cores used in the experiment.

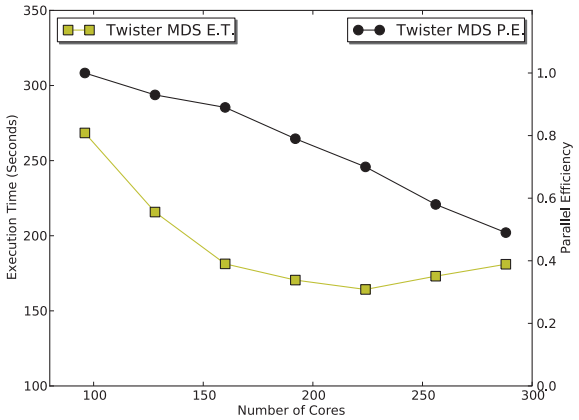

Figure 7. Twister MDS Execution Time and Parallel Efficiency

However, the parallel efficiency drops greatly once the number of cores increases (See Figure 7.). Besides, the execution time even grows at some point. The reason is that the cost of data broadcasting increases as the number of cores grows. For example, in the case that 288 cores are used, more than half of the execution time is used in data transmission. Though the communication burden of broadcasting data is due to the algorithm requirement and the problem can be eased by using more than one broker, this shows the limitation of one message broker and broadcasting data through broker should be carefully used in Twister iterative application design.

5. Conclusions and Future Work

In this paper, we present four parallel applications: Twister BLAST, Twister MDS Interpolation, Twister GTM Interpolation, and Twister MDS, with their implementations and performance measurement. We show that Twister can be applied not only on applications with non-iterative MapReduce programming model, but also on iterative MapReduce programming model. New applications extend the scope of applications using Twister. With iterative MapReduce functions, data partitioning, caching and reusable configuration, Twister can solve problems in a flexible and efficient fashion.

As a runtime of iterative MapReduce, Twister aims to provide functionalities to accelerate the computation of iterative algorithms. However, it is limited by the availability of messaging middleware. Though having open interface to messaging software is a good property, its performance largely depends on the performance of messaging middleware adopted. For instance, according to MDS iterative algorithm, the amount of broadcasting messages of temporary results between iterations is so large that certainly influences the messaging performance. This brings an interesting research issue of balancing the requirement of iterative algorithm and the capability of messaging middleware. Twister scripts can simulate some functions of distributed file systems but needs further optimization. In future work, we will integrate Twister with a customized messaging middleware and a distributed file system.

6. References

[1] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters.* Commun. ACM, 2008. **51**(1): p. 107-113.

[2] J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox., *Twister: A Runtime for iterative MapReduce*, in *Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010*. 2010, ACM: Chicago, Illinois.

[3] Apache, *Apache Hadoop*, Retrieved April 20, 2010, from ASF: http://hadoop.apache.org/core/.

[4] Amazon, *Amazon Web Services*.http://aws.amazon.com/.

[5] J. B. MacQueen. *Some Methods for classification and Analysis of Multivariate Observations.* in *5-th Berkeley Symposium on Mathematical Statistics and Probability*: University of California Press.

[6] S. Brin and L. Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine*; Available from: http://infolab.stanford.edu/~backrub/google.html.

[7] NaradaBrokering. *Scalable Publish Subscribe System*, 2010 [accessed 2010 May]; Available from: http://www.naradabrokering.org/.

[8] Apache, "ActiveMQ," http://activemq.apache.org/, 2009.

[9] NCBI. *BLAST*, 2010; Available from: http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastNews#1.

[10] NCBI. *BLAST Command Line Applications User Manual*, 2010; Available from: http://www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=helpblast&part=CmdLineAppsManual.

[11] George Coulouris Christiam Camacho, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer and Thomas L Madden, *BLAST+: architecture and applications.* BMC Bioinformatics 2009, 10:421, 2009.

[12] PolarGrid. *Cyberinfrastructure for Polar Expeditions*, 2010 [accessed 2010 January]; Available from: http://www.polargrid.org/polargrid/index.php/Main_Page.

[13] NCBI. *Databases available for BLAST search*; Available from: http://www.ncbi.nlm.nih.gov/blast/blast_databases.shtml.

[14] Darling A, Carey L, and Feng WC, *The Design, Implementation, and Evaluation of mpiBLAST.* In: Proc ClusterWorld, 2003. **2003**.

[15] A. Matsunaga, M. Tsugawa, and J. Fortes. *CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications*. in *IEEE Fourth International Conference on eScience (eScience '08)*. 2008. Indianapolis, IN.

[16] Wei Lu, Jared Jackson, and Roger Barga, *AzureBlast: A Case Study of Developing Science Applications on the Cloud*, in *ScienceCloud: 1st Workshop on Scientific Cloud Computing co-located with HPDC 2010 (High Performance Distributed Computing)*. 2010, ACM: Chicago, IL.

[17] "MPI," *Message Passing Interface*, http://www-unix.mcs.anl.gov/mpi/, 2009.

[18] NCBI, *NCBI Toolkit.*http://www.ncbi.nlm.nih.gov/BLAST/developer.shtml

[19] Pavan Balaji Heshan Lin, Ruth Poole, Carlos Sosa, Xiaosong Ma and Wu-chun Feng, *Massively Parallel Genomic Sequence Search on the Blue Gene/P Architecture*, in *SC2008*. 2008.

[20] *Hadoop Distributed File System HDFS*, 2009 [accessed 2009 December]; Available from: http://hadoop.apache.org/hdfs/.

[21] *Windows Azure Platform*, Retrieved April 20, 2010, from Microsoft: http://www.microsoft.com/windowsazure/.http://www.microsoft.com/windowsazure/.

[22] *ElimDupes*; Available from: http://hcv.lanl.gov/content/sequence/ELIMDUPES/elimdupes.html.

[23] *geneious*; Available from: http://www.geneious.com/default,1266,new_features.sm.

[24] Victor Seguritan and Forest Rohwer, *FastGroup: A program to dereplicate libraries of 16S rDNA sequences.* BMC Bioinformatics, 2001. **2:9**.http://www.ncbi.nlm.nih.gov/pmc/articles/PMC59723/

[25] Joshua Bulgrien David Kerk, Douglas W. Smith, Brooke Barsam, Stella Veretnik, and Michael Gribskov, *The Complement of Protein Phosphatase Catalytic Subunits Encoded in the Genome of Arabidopsis.* Plant Physiology, 2002. **129**: p. 908–925.

[26] Doris M. Kupfer Scott D. Drabenstot, James D. White, David W. Dyer, Bruce A. Roe, Kent L. Buchanan and Juneann W. Murphy, *FELINES: a utility for extracting and examining EST-defined introns and exons.* Nucleic Acids Research, 2003. **31**.

[27] *Center for Genomics and Bioinformatics*; Available from: http://cgb.indiana.edu/.

[28] J. B. Kruskal and M. Wish, *Multidimensional Scaling*. 1978: Sage Publications Inc.

[29] Michael W. Trosset and Carey E. Priebe, *The Out-of-Sample Problem for Classical Multidimensional Scaling.* 2006, Bloomington, IN: Indiana University.

[30] Ingwer Borg and Patrick J Groenen, *Modern Multidimensional Scaling: Theory and Applications.* 2005: Springer.

[31] Jong Youl Choi Seung-Hee Bae, Judy Qiu, Geoffrey C. Fox, *Dimension Reduction and Visualization of Large High-dimensional Data via Interpolation*, in *HPDC'10* 2010: Chicago, Illinois USA.

[32] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox, *Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications*, in *Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference*. 2010: Chicago, Illinois.

[33] Indiana University Bloomington Open Sysem Lab. *MPI.NET*, 2008; Available from: http://osl.iu.edu/research/mpi.net/.

[34] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, *Dryad: Distributed data-parallel programs from sequential building blocks*, in *ACM SIGOPS Operating Systems Review*. 2007, ACM Press. p. 59-72.

[35] NCBI. *PubChem*; Available from: http://pubchem.ncbi.nlm.nih.gov/.

[36] Christopher M. Bishop, Markus Svensén, and Christopher K. I. Williams, *GTM: The generative topographic mapping.* Neural computation, 1998. **10**: p. 215--234.

[37] M. Carreira-Perpinan and Z. Lu. *The Laplacian eigenmaps latent variable model*. in *11th Int. Workshop on Artifical Intelligence and Statistics*. 2007.

[38] A. Kaban. *A scalable generative topographic mapping for sparse data sequences*. in *the International Conference on Information Technology: Coding and Computing*. 2005.

[39] F. Nie S. Xiang, Y. Song, C. Zhang and C. Zhang, *Embedding new data points for manifold learning via coordinate propagation.* Knowledge and Information Systems, 2009. **19(2)**: p. 159-184.

[40] J. Kruskal, *Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis.* Psychometrika, 1964. ***29***: p. 1-27.