Distributed Scheduling and monitoring service leveraging FaBRiQ as a building block for CloudKon+

Arvind Shekar, Arihant Raj Nagarajan, Itua Ijagbone, Shivakumar Vinayagam
A20328640, A20334121, A20337217, A20341139
{anln43, anagara6, iijagbon, svinayag}@hawk.iit.edu,

***Abstract*—In today's world, the scientific community is moving towards distributed systems which plays an important role on achieving good performance and scalability. Task scheduling and execution over large scale, distributed systems plays an important role on achieving good performance and high system utilization[15]. Most of todays state-of-the-art job execution systems are centralized architectures, which have inherent limitations, such as scalability issues at extreme scales and single point of failures. On the other hand distributed job management systems are complex, and employ non-trivial load balancing algorithms to maintain good utilization. Thus we propose a distributed task execution framework which will provide the Load Balancing inherently using a distributed message passing Interface which is essentially a Distributed queue.**

CloudKon+ is a distributed task execution framework that can support distributed HPC[12] and MTC[15] scheduling, running millions of tasks on multiple nodes. It is built on FaBRiQ[2] which is a distributed message passing interface with high system utilization and scalability. The goal in this project is to enhance existing CloudKon[1] by replacing the Amazon SQS[18] Queue and Amazon DynamoDB[19] with FaBRiQ[2] and ZHT[3] which makes it possible to run on any public or private cloud infrastructure, Supercomputers, Science Grids[7] etc. We also propose an independent Monitoring framework which provides us several ways of knowing what the workers are up to, helping us to find potential bottlenecks in the process.

I. Introduction

The goal of a Distributed Task Scheduling System is to utilize the computing power of supercomputers, workstations and Distributed Systems present across the internet and maximize the system utilization and performance of the system. Hence it is predicted that exascale systems will come into being in the next decade. With the dramatic increase of the scales of today's distributed systems, it is urgent to develop efficient job schedulers. There are various distributed systems in the current world such as Hadoop[29],sparrow[26], slurm[25], condor[4],PBS[5], SGE[6]. Most schedulers have centralized Master/Slaves architecture,where a centralized server is in charge of the resource provisioning and job execution. Most have poor scalability at the extreme scales of petascale systems[8] with fine-granular workloads. The solution to this problem is to move to the decentralized architectures[5][4] that avoid using a single component as a manager. Distributed schedulers are normally implemented in either hierarchical or fully distributed architectures to address the scalability issue. The problem with centralized scheduler is that it will have low performance and system utilization for petascale systems[8] with fine granularity workloads.

Only limited systems support HPC[12] workload which can be performed on supercomputers and Grid systems. Hence we have rebuilt CloudKon[1] in C++ in order to integrate with FaBRiQ[2] and ZHT[3] and also to support HPC[12] workloads. Hence the Proposed system can also be built upon both private or public Cloud infrastructure, supercomputers and grids[7]. Task execution framework are not new in the distributed computing area. They have been around for quite a long time and have played a major role in distributing the task. Currently MATRIX[30] has similar architecture as CloudKon[1] and CloudKon+, except that CloudKon[1] focuses on the Cloud environment, and relies on the Cloud services, SQS[18] to do distributed load balancing, and DynamoDB[19] as the DKVS to keep task metadata while MATRIX focuses on Many-Tasking Computing where it uses DAGs.

As Cloudkon[1] is platform dependent and can run only on Amazon cloud infrastructure we have been restricted to use Amazons services such as SQS[18] and DynamoDB[19]. Also, Cloudkon[1] is implemented in Java making use of Amazon APIs. In this paper, we design and implement a scalable task execution framework that is platform independent and can be used in both public and private cloud infrastructure, and aimed at supporting both many-task computing and high-performance workloads.

II. Related Work

MATRIX[30] a distributed task execution fabric which adopted the adaptive work stealing for distributed load balancing and a distributed key-value store for task metadata management. The work stealing from neighbours could lead to a bottleneck.

Falkon [9] have centralized master/slaves architecture where a controller is handling all the activities, such as metadata management, resource provisioning, and job submission. This centralized architecture is not suited for the demands of exascale computing, due to both poor scalability and single point-of-failure.

Sparrow[26] is a distributed task scheduler with multiple schedulers pushing tasks to workers. Each scheduler knows all the workers. When dispatching a batch of tasks, a scheduler probes multiple workers (based on the number of tasks), and pushes tasks to the least overloaded ones. Once the tasks are submitted to a worker, they cannot be migrated in case of load

imbalance.

CloudKon[1] is a compact and lightweight distributed task execution framework that runs on the Amazon Elastic Compute Cloud (EC2) [17] by leveraging complex distributed building blocks such as the Amazon Simple Queuing Service (SQS) [18] and the Amazon distributed NoSQL key/value store (DynamoDB) [19].Condor [4] was implemented to harness the unused CPU cycles on workstations for long-running batch jobs.

Slurm [25] is a resource manager designed for Linux clusters of all sizes. It allocates exclusive and/or non-exclusive access to resources to users for some duration of time so they can perform work, and provides a framework for starting, executing, and monitoring work on a set of allocated nodes.

Portable Batch System (PBS) [5] was originally developed at NASA Ames to address the needs of HPC[11], which is a highly configurable product that manages batch and interactive jobs, and adds the ability to signal, rerun and alter jobs. LSF Batch [20] is the load-sharing and batch-queuing component of a set of workload management tools.

All these systems target as the HPC[12] or HTC applications, and lack the granularity of scheduling jobs at finer levels making them hard to be applied to the MTC[15] applications. Whats more, the centralized dispatcher in these systems suffers scalability and reliability issues. Hence we can conclude that Distributed Systems are more scalable with the ability to handle low granularity tasks at high Frequency. Thus Distributed systems like Matrix and sparrow can run at a higher scale and give significantly high throughput. But the Task Execution Framework systems like Cloudkon and Cloudkon+ can give a better performance than Distributed schedulers as the load balancing is done inherently.

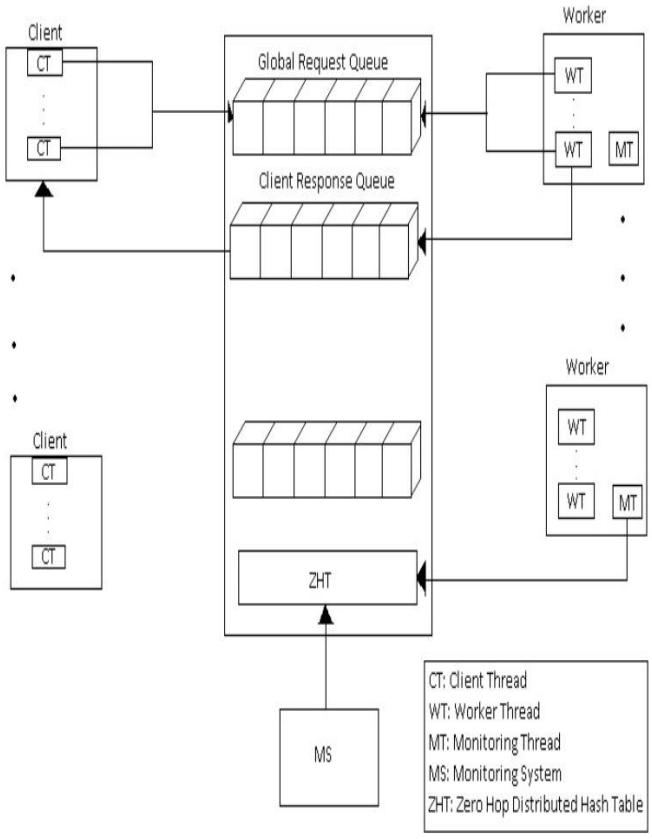

Fig. 1. CloudKon+ architecture overview.

III. System Architecture

A. CloudKon+

Build on top of FaBRiQ[2] a distributed MTC[15] scheduler (called CloudKon+), similar to that of CloudKon[1] (with the exception that it will be platform independent of any cloud). In cloudkon[1], the distributed message queue used is SQS[18] (Simple Queuing Service) which is provided by Amazon. Since SQS[18] only assures at least once delivery of the message, this leads to duplicate messages being sent to more than one worker at the same time. This problem was overcome in Cloudkon[1] by using DynamoDB[19] to map each message retrieved by the workers and since DynamoDB[19] was atomic in nature it prevent duplicate messages. Hence we propose a solution called cloudkon+ ,which replaces the SQS[18] and DynamoDB[19] setup in cloudkon[1] with FaBRiQ[2] which is also a distributed message queue which assures exactly once delivery. In order to achieve the best performance, and for your system to run in most distributed systems environments, CloudKon+ needs to be written in C++, since FaBRiQ[2] is in C++.The main components of the CloudKon+ for running MTC[15] jobs are Client, Worker, Global Request Queue implemented on FaBRiQ[2] and the Client Response Queues which are implemented using TCP sockets.

The underlying structure of FaBRiQ[2] queue,is that each message is presented as a key value pairs in ZHT[3]. The ZHT[3] maintains a global Hash Table which stores the values under the key. ZHT[3] uses metadata table which is synchronized across the other nodes in a ring fashion. The ZHT[3] nodes communicate with each other using TCP connection and google protocol buffer to serialize the data send it across the network. Hence the message inserted will be placed on a random node and this location can be mapped using the metadata table. Message hopping is done by using an algorithm which selects the messages from that node first and if not present then selects a random node to retrieve from using the metadata table till it gets a message. The algorithm finds an message based on randomized search which gives a average hop count of 3 for few messages.Our system architecture will contain a client which pushes messages on the FaBRiQ[2] global request queue. The messages would contain the details of the client along with the work to be executed and the worker would parse the data that was popped from FaBRiQ[2]. The worker would process the data and send a message back to the client identifying it with client ip and using a socket TCP connection. The client will run a background thread which keeps listening in the socket port for the messages from the worker to get the status of the task done. Hence the client will map the number of tasks sent to the messages received.

B. Monitoring System

An independent monitoring system which will monitor the states of each worker node and the entire system as a whole. Each worker through an instance has the ability to send its own stream of data. Because we wanted encapsulation as much as we could get, we abstracted our client api. When a worker calls our client api to send its data, we create a separate thread to execute this process. This is done so we dont block the worker node from doing what it was made to do. Our underlying monitoring system uses Apache Kafka[32], a distributed messaging system. With a Connector (Consumer process), we can consume and process data from Apache Kafka[32]. Connectors are written by those who need them. We are working towards making the monitoring system as independent as possible by allowing owners of workers define whatever data they need and writing Connector services to consume and process the data. Since we know that we would eventually run out of memory because of the many writes, Apache Kafka[32] includes a policy where owners define how long they want their data to remain in memory before pushing them to some form of external storage.

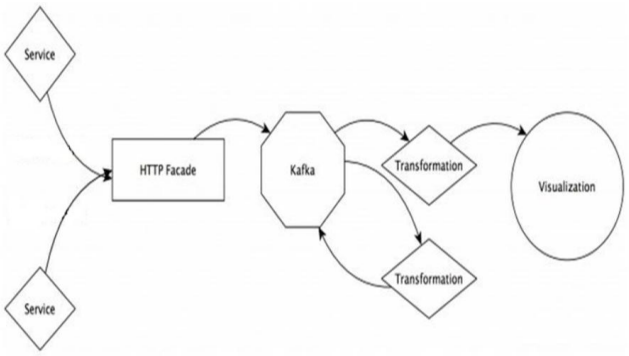

Fig. 2. CloudKon+ architecture overview.

Our currently design architecture due to time constrain, collocates the web server and Apache Kafka on the same node. This however doesnt show any form of lag on our system as can be seen from our latency evaluation. We went for a web server approach to receiving data mainly because our Workers are written in the c/c++ language. We also didnt want to tie down our workers to be dependent on Apache Kafka, in case the api for Kafka changes. Instead we choose to go with a schema. Owners define a schema using Apache Avro, they then collect data that meets this schema requirements and send it over the network to our web server. Our worker client API does exactly this. When a worker takes a task to process, we mark the state as 1, when are worker task is completed, we mark the worker state as 0. We have explained how we have used this state changed. On the other hand our Connectors run on different node.

C. Kafka Architecture

We needed to use a stream based processing approach[17,18]. Using a stream based approach required that our data had to be immutable and as a result had to come in order. We decided to use Apache Kafka[32]. Apache Kafka[32]

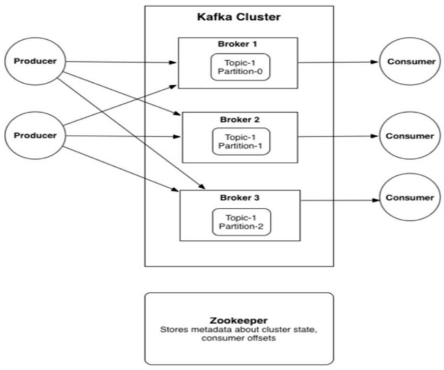

Fig. 3. Throughput.

is a distributed system designed to handle streams. It is built to be fault-tolerant, high-throughput, horizontally scalable, and allows geographically distributing data streams and processing. Kafka[32] is a structured commit log, which solves the need for ordered messages

IV. System Deployment

The System can be deployed by downloading the source code from . Use the CloudKon+ Installation. The README.md file included with in the file and step wise instruction as to install and run the files are given. First we need to start up the ZHT[3] which is the base process for both FaBRiQ[2] and FusionFS, which in turn will also be started. Then if the monitoring of the nodes is required, we start it up next as a seperate cluster of Kafka[32] and tomcat server. We then start up the monitoring node which observe the whole system. Finally we start up the workers and clients of the CloudKon+ to send and receive the messages.

V. Evaluation

A. Testbed

We deployed and ran CloudKon+ on Amazon EC2[17] instances. We have used t2.medium instances on Amazon EC2[17] as the virtual machine contains 4 hardware threads and it is the most balanced instance. We have run all of our experiments on us.west.2 datacenter of Amazon. We have scaled the experiments from 1 up to 16 nodes. In order to make the experiments efficient, client and worker nodes both run on same node. All of the instances had Ubuntu 14.04 Operating Systems. Our framework works on any OS that has a JRE 1.7,Openssh,Amazon AWS console,G++,GCC,Protobuf- 2.4.1 and protobuf-c-0.15,. We have used Bash scripting language for parallel ssh to start ,stop and execute the servers in each node simultaneously, file transfer from EC2[17] instances, Parallel-SSH for parallel execution of client and server code on EC2[17] instances , get EC2[17] IP address, etc.,

B. Throughput

In order to measure the throughput of our system we run sleep 0 tasks. There are 1 worker threads running on each instance. The client submits about 10000 tasks to the FaBRiQ[2] queue.Figure (number) provides the throughput of CloudKon+ on different scales. Each instance will run a worker process

and a FaBRiQ[2] process. Thus there will be a overhead of the FaBRiQ[2], ZHT[3], monitoring process,Cloudkon+ client and worker on each node. Also as the no of task remaining in the FaBRiQ[2] is decreasing, the pop of a message finds it hard to retrieve a message as only a few remain in the system and those can be found using only multiple hops causing a delay for each pop.

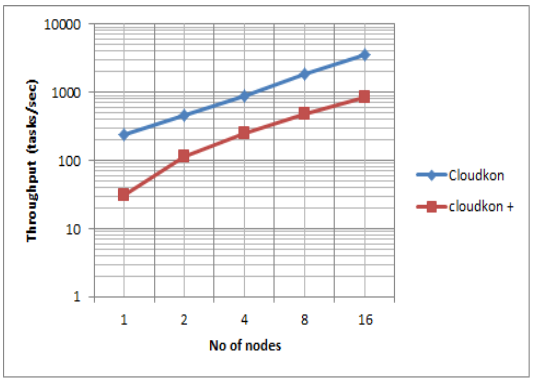

Fig. 4. Throughput-Diff workload range

As we can see from the figure 4 that cloudkon+ performs in the same scale as the cloudkon. They both are parallel lines and the difference in their throughput is due a few differences. Cloudkon was tested using c3.large instances as test nodes while we used t2.medium as the test nodes. Also cloudkon was given a scaling workload of tasks which went from 16000 to 1.38 million tasks. But cloudkon+ on the other hand was tested with a fixed workload of tasks. We can see that the throughput difference is also due to the fact that there is a high overload in cloudkon+ as FaBRiQ,ZHT,Cloudkon+ client and Monitoring system are running on the same system hence using up cpu cycles while compared to cloudkon's SQS and dynamoDB which are seperate from the system.

Hence we tried to compare again (figure 5) with other systems like matrix having the same workload range and using the same instance type. We found that the system performed better at the same range. Hence the Cloudkon+ system scales properly as the no of nodes increasely, thus it distributes evenly and the overload also increases gradually but compared to the overall system workload it is low. Hence cloudkon+ can perform better than matrix and cloudkon at small scales of the same workload. We used a simple cloudkon implementation using just a push and pop system using SQS and dynamoDb without any threading or batching. Matrix was implemented in a similar manner.

C. Efficiency

We tested the system efficiency in case of both homogeneous tasks. The homogeneous tasks have a certain task duration length. Therefore it is easier distribute them since the scheduler assumes it takes the same time to run them. This could give us a good feedback about the efficiency of the system in case of running different task types with different granularity. We can also assess the ability of the system to run the very shot length tasks.In this section we evaluate the efficiency of CloudKon+ second tasks. As shown in the

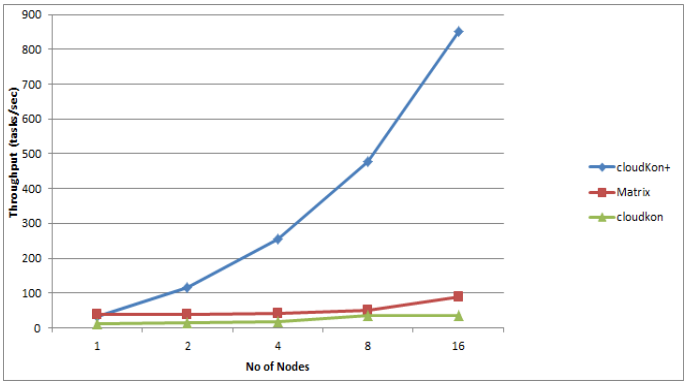

Fig. 5. Throughput-Same workload range

graph(figure 6) the Cloudkon+ system performs with high efficiency for small no of worker as the system overhead is small. But as the no of workers increases ,the efficiency decreases as the overhead of maintaining FaBRiQ[2] and ZHT[3] in the same systems as the worker increases drastically. The no of TCP connections increases exponentially as the no of nodes in server increase ,since the cost of maintaining a distributed synchronous state across the system is high. Also as the no of task remaining in the FaBRiQ[2] is decreasing, the pop of a message finds it hard to retrieve a message as only a few remain in the system and those can be found using only multiple hops causing a delay for each pop.

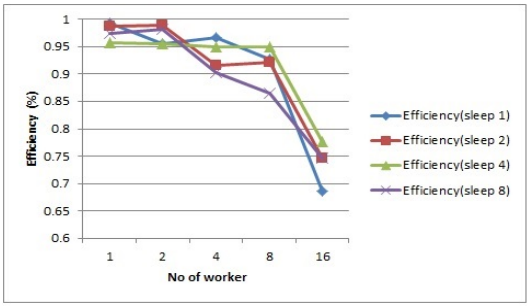

Fig. 6. Efficiency

D. Monitoring System - Latency

The Monitoring System was setup to monitor the workers in the wordcount implementation of CloudKon+, through this we were also able to calculate the average latency from the workers to the monitoring system and back for a scale of 1 to 2 workers with a file sizes from 5, 10, 15, 20 and 25 MB respectively. The worker when it retrieves a task from the FaBRiQ it sends a message to the monitoring system. The monitoring system after processing the message sends back a confirmation which is then timed by the worker. Hence we have plotted a graph (figure 7) which maps the up and down communication latency of the system.

VI. Use Case Implementation

A. Wordcount

We have implemented the wordcount use case in a style similar to swift using cloudkon+ with the help of FusionFS

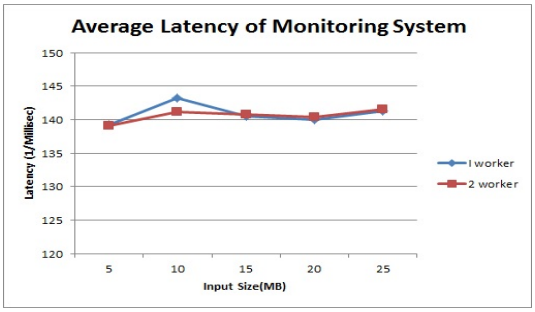

Fig. 7. Latency

as a distributed storage system for varying input sizes of 1,5,10,15,20 and 25 MB. The client breaks the input into chunks of 100 kb and stores it under fusion_mount point created by FusionFS. Then the client maps the split files locations and sends it to the worker through the FaBRiQ[2], which in turn reads the input and then does wordcount using a Hashmap implementation outputting it to a intermediate output file. Once the tasks are completed client is notified which merges it and sorts the result. Hence we built a use case implementation of our system by adding FusionFS as a distributed storage and ran a Wordcount program on it and benchmarked its Throughput (figure 8).

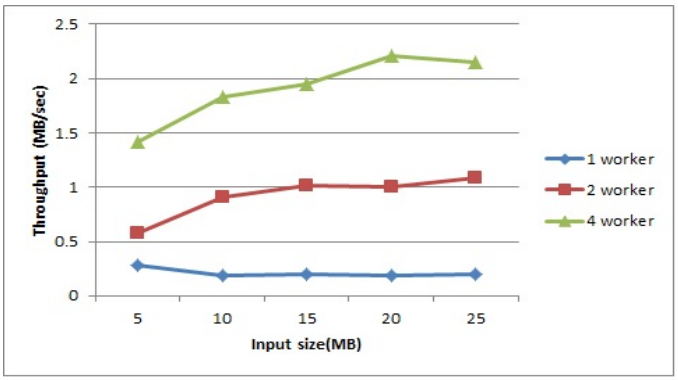

Fig. 8. Word-Count throughput

B. Worker State - Monitoring

As one of our use cases, we wanted to monitor the heart rate (figure 9) of our workers. We implemented a Collector service that consumes streams of data from Apache Kafka which have have been provided by the workers. As stated earlier, when a client API takes a task to process we send the current state of the worker (a 1) to the monitoring system. When the worker is done completing a task, we again send the current state of the worker (a 0) to the monitoring system. On the other end, the Collector takes the data computes on it and sends it to file.

VII. Future Work

Implement scientific HPC[12] and MTC[15] use cases using Cloudkon+ and then benchmark it to observe the performance. Implement Cloudkon+ to use DAG to map the task dependencies.

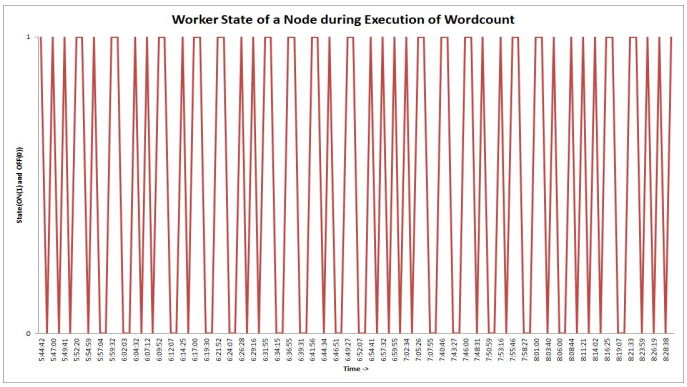

Fig. 9. Monitoring of Worker State

Implement the monitoring system to use RRDtools to generate and print graphs automatically and thus create a versatile system.

VIII. Conclusion

We learned a lot about distributed task scheduling systems. We understood the various bottlenecks and hurdles in distributed systems and also built a new Cloudkon+ system which implements a task execution framework. We have also implemented a independent monitoring system which collect the monitoring data needed to plot graphs which can be coupled with any Distributed system. Hence we have learned the working behind Cloudkon[1], FaBRiQ[2],Kafka[32],FusionFS and ZHT[3]. We also became familiar with the Amazon Aws console , problems in distributed system, multithreading in c++ and java,developing a scalable and efficient code suitable for distributed system, debugging a highly distributed and multi-threaded code, benchmarking at high scales, shell scripting and shell commands for parallel ssh. With Cloudkon+, we were able to build a system which can be deployed in both HPC[12],MTC[15] and Cloud environments. With monitoring system, we built a versatile system which can be integrated with any existing distributed system.

References

[1] Iman Sadooghi, Ioan Raicu. "CloudKon: a Cloud enabled Distributed tasK executiON framework", Illinois Institute of Technology, Department of Computer Science, PhD Oral Qualifier, 2013

[2] Iman sadooghi, Ke Wang, et al,FaBRiQ: Leveraging Distributed Hash Tables towards Distributed Publish-Subscribe Message Queues

[3] T. Li, et al., ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table, in IEEE International Parallel a Distributed Processing Symposium, IEEE IPDPS 13, 2013.

[4] D. Thain, T. Tannenbaum, M. Livny, Distributed Computing in Practice: The Condor Experience Concurrency and Computation: Practice and Experience 17 (2-4), pp. 323-356, 2005.

[5] B. Bode et. al. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters, Usenix, 4th Annual Linux Showcase and Conference, 2000.

[6] W. Gentzsch, et. al. Sun Grid Engine: Towards Creating a Compute Power Grid, 1st International Symposium on Cluster Computing and the Grid, 2001.

[7] C. Dumitrescu, I. Raicu, I. Foster. Experiences in Running Workloads over Grid3, The 4th International Conference on Grid and Cooperative Computing (GCC 2005)

[8] I. Raicu, et. al. Toward Loosely Coupled Programming on Petascale Systems, IEEE SC 2008.

[9] I. Raicu, et. al. Falkon: A Fast and Light-weight tasKexecutiON Framework, IEEE/ACM SC 2007.

[10] S. Melnik, A. Gubarev, J. J. Long, G. Romer,S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive Analysis of Web-Scale Datasets. Proc. VLDB Endow., 2010.

[11] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. 2012. Performance evaluation of Amazon EC2 for NASA HPC applications. In Proceedings of the 3rd workshop on Scientific Cloud Computing (ScienceCloud '12). ACM, NY, USA, pp. 41-50.

[12] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn. Case study for running HPC applications in public clouds, In Proc. of ACM Symposium on High Performance Distributed Computing, 2010.

[13] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In IEEE INFOCOM, 2010.

[14] P. Kogge, et. al., Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[15] I. Raicu, Y. Zhao, I. Foster, Many-Task Computing for Grids and Supercomputers, 1st IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2008.

[16] I. Raicu. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", Computer Science Dept., University of Chicago, Doctorate Dissertation, March 2009

[17] Amazon Elastic Compute Cloud (Amazon EC2), Amazon Web Services,

[18] 2013, http://aws.amazon.com/ec2/

[19] Amazon SQS,2013, http://aws.amazon.com/sqs/

[20] Amazon Dynamo DB http://aws.amazon.com/dynamodb/

[21] LSF:http://platform.com/Products/TheLSFSuite/Batch, 2012.

[22] L. V. Kaleet. al. Comparing the performance of two dynamic load distribution methods, In Proceedings of the 1988 International Conference on Parallel Processing, pages 811, August 1988.

[23] W. W. Shu and L. V. Kale, A dynamic load balancing strategy for the Chare Kernel system, In Proceedings of Supercomputing 89, pages 389398, November 1989.

[24] A. Sinha and L.V. Kale, A load balancing strategy for prioritized execution of tasks, In International Parallel Processing Symposium, pages 230237, April 1993.

[25] M.H. Willebeek-LeMair, A.P. Reeves, Strategies for dynamic load balancing on highly parallel computers, In IEEE Transactions on Parallel and Distributed Systems, volume 4, September 1993

[26] M. A. Jetteet. al, Slurm: Simple linux utility for resource management. In In Lecture Notes in Computer Sicence: Proceedings of Job Scheduling Strategies for PrarallelProcesing (JSSPP) 2003 (2002), Springer-Verlag, pp. 44-60.

[27] OUSTERHOUT, K., WENDELL, P., ZAHARIA, M., AND STOICA, I. Sparrow: Scalable scheduling for sub-second parallel jobs. Tech. Rep. UCB/EECS-2013-29, EECS Department, University of California, Berkeley, Apr 2013.M.

[28] Frigo, et. al, The implementation of the Cilk-5 multithreaded language, In Proc. Conf. on Prog. Language Design and Implementation (PLDI), pages 212223. ACM SIGPLAN, 1998.

[29] R. D. Blumofe, et. al. Scheduling multithreaded computations by work stealing, In Proc. 35th FOCS, pages 356368, Nov. 1994.

[30] Apache Hadoop ,https://hadoop.apache.org/

[31] V. Kumar, et. al. Scalable load balancing techniques for parallel computers, J. Parallel Distrib. Comput., 22(1):6079, 1994.

[32] Anupam Rajendran, Ioan Raicu. "MATRIX: Many-Task Computing Execution Fabric for Extreme Scales", Department of Computer Science, Illinois Institute of Technology, MS Thesis, 2013

[33] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. NetDB, 2011.

[34] J. Dinanet. al. Scalable work stealing, In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09), 2009.

[35] Tonglin Li, Raman Verma, Xi Duan, Hui Jin, Ioan Raicu. Exploring Distributed Hash Tables in High-End Computing, ACM Performance Evaluation Review (PER), 2011

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, Spark: Cluster Computing with Working Sets, in Proceedings of the 2nd USENIX Conference on Hot topics in Cloud Computing, (Boston,MA), June 2010.

[37] I. Raicu, I. Foster, et. al. The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems, ACM HPDC 2009.

[38] Stream processing http://blog.confluent.io/2015/01/29/making-sense-of-stream-processing/