2nd IEEE International Conference on Cloud Computing Technology and Science

CloudBATCH: A Batch Job Queuing System on Clouds with Hadoop and HBase

Chen Zhang
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
Email: c15zhang@cs.uwaterloo.ca

Hans De Sterck
Department of Applied Mathematics
University of Waterloo
Waterloo, Canada
Email: hdesterck@math.uwaterloo.ca

***Abstract*—As MapReduce becomes more and more popular in data processing applications, the demand for Hadoop clusters grows. However, Hadoop is incompatible with existing cluster batch job queuing systems and requires a dedicated cluster under its full control. Hadoop also lacks support for user access control, accounting, fine-grain performance monitoring and legacy batch job processing facilities comparable to existing cluster job queuing systems, making dedicated Hadoop clusters less amenable for administrators and normal users alike with hybrid computing needs involving both MapReduce and legacy applications. As a result, getting a properly suited and sized Hadoop cluster has not been easy in organizations with existing clusters. This paper presents CloudBATCH, a prototype solution to this problem enabling Hadoop to function as a traditional batch job queuing system with enhanced functionality for cluster resource management. With CloudBATCH, a complete shift to Hadoop for managing an entire cluster to cater for hybrid computing needs becomes feasible.**

I. Introduction

MapReduce [1] is becoming increasingly popular and used to solve a wide spectrum of problems ranging from bioinformatics to astronomy. The increase of MapReduce usage requires a growing number of Hadoop [2] clusters. However, because Hadoop is incompatible with existing cluster batch job queuing systems and requires a dedicated cluster under its full control, it is not easy to get a Hadoop cluster within organizations with existing cluster resources due to administrative difficulties. It is also difficult to manage Hadoop clusters due to Hadoop's lack of functionality for job management, user access control, accounting, fine-grain performance monitoring, etc. As a result, Hadoop clusters may suffer from usage contention/monopoly if shared by multiple user groups. Additionally, even for institutions that can afford several dedicated clusters for various usages, provisioning resources for Hadoop clusters is still tricky since the resources, once dedicated to Hadoop, cannot be used for the load balancing of legacy batch job submissions to other overloaded traditional clusters even if they are idle. This is because Hadoop is designed for running a certain type of applications, mainly back-end server computations such as page rank calculations, which are normally MapReduce programs for large-scale data processing tasks rather than general computing problems with diverse types of legacy programs and paradigms.

Going for public cloud services such as Amazon EC2 and S3 is not suitable for all Hadoop needs either. Specifically, it could be too expensive or infeasible to transfer and store large amounts of data with Amazon. For example, a live cell image processing application described in [7] generates more than 300GB of data daily. Assuming 3MB/s WAN bandwidth, it would take about 27 hours to transfer all the data into Amazon at the cost of $30[1], and another $45 per day for storage if using Amazon S3. Worse still, some of these big data sets might only need to be processed once and then discarded or archived, making the expensive data transfer even less cost-effective. It could be more suitable for these types of data processing tasks to be completed within the institutions where the data are generated.

Efforts are made to make Hadoop and traditional queuing systems coexist and work together. But these efforts have limitations themselves. The Hadoop project itself has tried to work in this direction and provides a Hadoop On Demand (HOD) package in support of provisioning and managing independent Hadoop MapReduce and HDFS instances on a shared cluster of nodes managed by a resource manager called "Torque". It is relatively easy to make HOD work with other resource managers such as Condor, SGE (Sun Grid Engine), and Moab. However, an important limitation [4] for HOD is that HOD cannot effectively move computation near data to exploit data locality because of its design choice of creating small sub-clusters for processing individual users' tasks in a subset of compute nodes where data may not reside. Other efforts have been made to develop new queuing systems to work with Hadoop. Oracle attempted to provide a solution and recently made a new release of the Oracle Grid Engine, SGE 6.2u5, claiming to be the first cluster resource management system with Hadoop integration so that users can submit Hadoop applications to an SGE cluster just like they would with any other parallel jobs. SGE will take care of setting up the Hadoop MapReduce on demand. Apart from the apparent limitation of requiring users to use/switch to SGE, the SGE Hadoop integration also suffers from the data locality problem, although less severe

[1]Inbound data transfer to Amazon is free till November 2010

978-0-7695-4302-4/10 $26.00 © 2010 IEEE
DOI 10.1109/CloudCom.2010.22

than HOD, as well as problems concerning non-exclusive host access (See Section II). Additionally, both HOD and SGE with Hadoop integration suffer from wasting resources during the reduce phase, which is an intrinsic drawback for dynamically creating a private Hadoop cluster per user MapReduce application request, and there exists no solution for this problem yet.

To address the above mentioned problems, this paper attempts to provide a solution where Hadoop can function as a traditional batch job queuing system with enhanced management functionalities in addition to its original resource management capability. The goal is to develop a system that enables cluster administrators to make a complete shift to Hadoop for managing an entire cluster for hybrid computing needs. Our system is named "CloudBATCH". The general idea about how CloudBATCH works is as follows: The system includes a set of HBase [3] tables that are used for storing resource management information, such as user credentials and queue and job information, which provide the basis for rich customized management functionalities. In addition, based on the HBase tables, our system provides a queuing system that allows "Job Brokers" to submit "Wrappers" (Map-only MapReduce programs) as agents to execute jobs in Hadoop. Users can submit jobs and check job status by using a "Client" at any time.

The remainder of the paper is structured as follows: in Section II we introduce related work about Hadoop schedulers, Hadoop on Demand, and SGE with Hadoop integration. In Section III we briefly describe some desired properties of a batch job queuing system for clusters. Section IV will describe the design and implementation of CloudBATCH and how CloudBATCH achieves the properties listed in Section III. In Section V we give a performance evaluation. Section VI concludes.

II. Related Work

In this section, we review existing solutions for Hadoop resource management and its integration with existing cluster batch job queuing systems, and examine their limitations. To our knowledge, none of the existing solutions can enable Hadoop with a rich set of traditional cluster management functionalities comparable to what CloudBATCH provides.

A. Hadoop Schedulers

Hadoop allows customized schedulers to be developed as pluggable components. The motivation for supporting various schedulers, besides an obvious objective of achieving better system utilization, originates from the user demands to share a Hadoop cluster allocated into different queues for various types of jobs (production jobs, interactive jobs, etc.) among multiple users. The default Hadoop scheduler is a simple FIFO queue for the entire Hadoop cluster which is insufficient to provide guaranteed capacity with fair resource allocations for mixed usage needs. Several schedulers are created for various purposes, such as the Capacity Scheduler, the Fair Scheduler [5], and the Dynamic Priority (DP) parallel task scheduler [4].

However, Hadoop schedulers cannot make Hadoop more favorable towards functioning as a cluster batch queuing system. They are task level schedulers that only concern stateless decisions for job executions and do not provide rich functionality for cluster level management, such as user access control and accounting, and advanced job and job queue management with easily customizable priority levels and reservation support.

B. Hadoop on Demand

As the Hadoop project's effort to alleviate its incompatibility with traditional queuing systems, Hadoop on Demand (HOD) is added to Hadoop for dynamically creating and using Hadoop clusters through existing queuing systems. The idea is to make use of the existing cluster queuing system to schedule jobs that each run a Hadoop daemon on a compute node. These running daemons together create an on-demand Hadoop cluster. After the Hadoop cluster is set up, user submitted MapReduce applications can be executed. A simple walkthrough of the process for creating a Hadoop cluster and executing user submitted MapReduce jobs through HOD is as follows:

1) User requests from cluster resource management system a number of nodes on reserve and submits a job called RingMaster to be started on one of the reserved nodes.
2) Nodes are reserved and RingMaster is started on one of the reserved nodes.
3) RingMaster starts one process called HodRing on each of the reserved nodes.
4) HodRing brings up on-demand Hadoop daemons (namenode, jobtracker, tasktracker, etc.) according to the specifications maintained by RingMaster in configuration files.
5) MapReduce jobs that were submitted by the user are executed on the on-demand cluster.
6) The Hadoop cluster gets torn down and resources are released.

As seen from the above walkthrough of an HOD process, data locality of the external HDFS is not exploited because the reservation and allocation of cluster nodes takes no account of where the data to be processed are stored, violating an important design principle and reducing the advantage of the MapReduce framework. Another problem with HOD is that the HodRing processes are started through ssh by RingMaster, and the cluster resource management system is unable to track resource usage and to perform thorough cleanup when the cluster is torn down.

C. SGE with Hadoop Integration

Recently, Oracle released SGE 6.2u5 with Hadoop integration, enabling SGE to work with Hadoop without requiring a separate dedicated Hadoop cluster. The core design idea is similar to HOD in that it also tries to start an on-demand Hadoop cluster by running Hadoop daemons through the cluster resource management system on a set of reserved compute nodes. The difference with HOD is that SGE exerts better concern for data locality when scheduling tasktrackers and supports better resource usage monitoring and cleanup because tasktrackers are directly started by SGE as opposed to be started by HodRing in HOD. The main problem, apart from locking users down to using SGE, lies in its mechanism of exploiting data locality and the non-exclusive usage of compute nodes.

Figure 1 illustrates how SGE with Hadoop integration works:

1) A process called Job Submission Verifier (JSV) talks to the namenode of an external HDFS to obtain a data locality mapping from the user submitted MapReduce program's HDFS paths to data node locations in blocks and racks. Note that the "Load Sensors" are responsible for reporting on the block and rack data for each execution host where an SGE Execution Daemon runs.
2) The SGE scheduler starts the Hadoop Parallel Environment (PE) with a jobtracker and a set of tasktrackers as near the data nodes containing user application input data as possible.
3) MapReduce applications that were submitted by the user are executed. Because several tasktrackers might have been started on the same physical node, physical nodes could be overloaded when user applications start to be executed.
4) The Hadoop cluster gets torn down and resources are released.

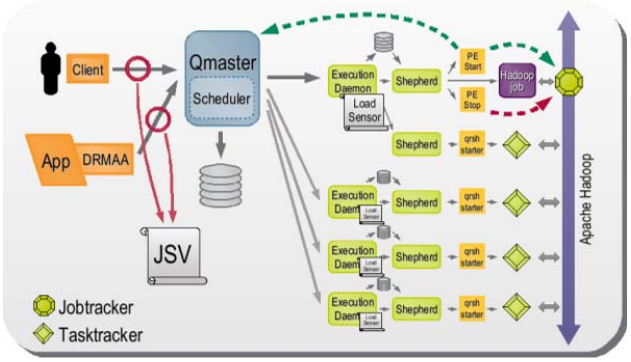

Figure 1. SGE Hadoop Integration (taken from online blog post by Oracle).

As seen above, the major advantage about SGE compared to HOD is the utilization of data locality information for scheduling tasktrackers. However, a compute node could get overloaded because multiple tasktrackers may be started on the same node for data locality concerns to execute tasks. In this case, an unpredictable number of Hadoop speculative tasks [5] may be started on the other idling nodes where data do not reside, incurring extra overhead in data staging and waste of resources for executing the otherwise unnecessary duplicated tasks. The unbalanced mingled execution of normal and speculative tasks may further mix up Hadoop schedulers built in with the dynamically created Hadoop cluster which are unaware of the higher level scheduling decisions made by SGE, harming performance in unpredictable ways. Even if the default Hadoop speculative task functionality is turned off, the potential danger of overloading a node persists, which could further contribute to unbalanced executions of Map tasks that result in wasting cluster resources as explained below. SGE also has an exclusive host access facility. But if this facility is required, then the data locality exploitation mechanism would be much less useful because normally a data node would potentially host data blocks needed by several user applications while only one of them can benefit from data locality in the case with exclusive host access. Additionally, the SGE installation file is about 700MB, quite large compared to Hadoop which is only about 70MB in a tar ball and thus less costly to distribute.

Worse still, both HOD and SGE suffer from the same major problem intrinsic to the idea of creating a Hadoop cluster on-the-fly for each user MapReduce application request. The problem is a possible significant waste of resources in the Reduce phase, where nodes might be idling when the number of Reduce tasks to be executed is much smaller than the number of Map tasks that are executed. This is because each of the on-demand clusters is privately tailored to a single user MapReduce application submission and the size of the Hadoop cluster is fixed at node reservation time. If the user MapReduce application requires far more Map tasks than Reduce tasks and the number of nodes are reserved matching the Map tasks (which is usually what users would request), many of the machines in the on-demand Hadoop cluster will be idling when the much smaller number of Reduce tasks are running at the end. The waste of resources could also occur in the case of having unbalanced executions of Map tasks, in which case a portion of Map tasks get finished ahead of time and wait for the others to finish before being able to enter the Reduce phase. There exists no solution for this problem yet.

III. Desired Properties

In this section, we describe some properties that Hadoop lacks good support of, yet are essnetial for batch job queuing systems. CloudBATCH supports the majority of these properties.

A. Job Queue Management

Job queue management is critical for managing shared usage of a cluster for various purposes among multiple users. For example, there could be a queue for batch serial jobs and a queue for MPI applications. Cluster nodes can be assigned to queues with a certain minimum and maximum quantity and capacity guarantee for optimized resource utilization. Additionally, each queue should optionally have specific policies concerning the limit of job running time, queue job priority, etc., to avoid the need to specify these policies on a per job basis. Furthermore, user access control for submitting to queues should also be in place.

B. Job Scheduling and Reservation

Jobs should be put into a certain queue with queue-specific properties, such as priority and execution time limit. Jobs with higher priorities should be scheduled first and may require preemption based on priority. Reservation for pre-scheduled job submission and customized job submission policies may be supported such as putting a threshold on the number of simultaneous job submissions allowed for each user. In addition, job status and history shall be kept on a per user basis together with descriptions for user management, statistical study and data provenance purposes. Resource status organized by queues shall also be maintained concerning activeness and workload. Both the status of jobs and resources shall be able to be monitored in real-time or offline for analysis.

C. User Access Control and Accounting

User access control shall be supported at least at the queue level, which requires user credentials to be kept and validated when a job submission or reservation is made. Stateful job execution status and history along with user information shall be kept to support accounting. User groups may also be supported.

In the following section, we discuss to which degree CloudBATCH succeeds in meeting these desired properties.

IV. CloudBATCH

A. Architecture Overview

CloudBATCH is designed as a fully distributed system on top of Hadoop and HBase. It uses a set of HBase tables globally accessible across all the nodes to manage Meta-data for jobs and resources, and runs jobs through Hadoop MapReduce for transparent data and computation distribution.

Figure 2 shows the architecture overview of the CloudBATCH system. CloudBATCH has several distributed components, Clients, Job Brokers, Wrappers and Monitors. The general sequence of job submission and execution using CloudBATCH is as follows: Users use Clients to submit jobs to the system. Job information with proper job status is put into HBase tables by the Client. In the meantime, a

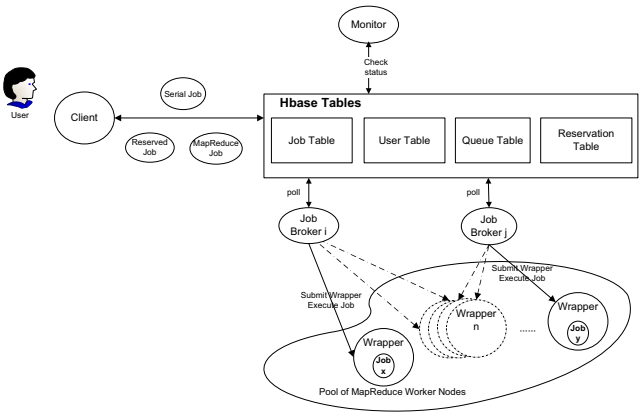

Figure 2. CloudBATCH architecture overview.

number of Job Brokers are polling for the HBase tables to find jobs ready for execution. If a job is ready, a Wrapper containing the job is submitted to Hadoop MapReduce by the Job Broker. The Wrapper is responsible for monitoring the execution status of the job and updating relevant records concerning job and resource information in HBase tables. The Wrapper is also responsible for following some job policies such as on execution time limit and may terminate a running job if it takes too long over the limit. Monitors are currently responsible for detecting and handling failures that may occur after Wrappers are submitted to Hadoop MapReduce but before they can start to execute the assigned jobs, which may result in jobs being marked as "Status:queued" but never getting executed.

The key part of CloudBATCH is a set of HBase tables that store Meta-data information for resources. The main tables are Queue Table (Table I), Job Table (shown in three parts in Tables II, III, and IV), Scheduled Time Table (Table V), and User Table (Table VI)[2].

The Queue Table (Table I) stores information about queues such as type of jobs (e.g., a queue for MapReduce jobs or serial jobs), queue capacity measured in the maximum number of simultaneous active jobs allowed, queue job priority (numerical values with increasing priority level from 1-10), execution time limit in milliseconds associated with the queue, queue domain (private for some users or public for all), and list of users or groups (access control list) allowed to access the queue. Note that Table I is a sparse table with rows of varying lengths. In our example settings, queue "serial" is under the "public" domain and does not specify a list of users/groups who are allowed to access the queue, while queue "bioinformatics" is under the "private" domain and only allows the listed users to submit jobs to it. There is no priority associated with queue "bioinformatics" meaning that either the default priority of "1" or the priority specified by the user at job submission will be used.

[2]HBase column families are omitted for simplicity for all tables.

The Job Table (Tables II, III, IV) is a large table containing extensive information about jobs. Here we show the essential columns only and is split into three tables for better display. Jobs are identified by unique IDs, submitted to queues, and associated with the submitting users. CloudBATCH currently accepts three types of jobs, namely, serial, MapReduce and Scheduled Time (i.e., serial or MapReduce jobs scheduled with an earliest start time). Each job has an associated priority (either inherited from the queue it is submitted to or specified explicitly through the Client), execution length limit, SubmitTime (the time at the HBase server when the Client first puts in the job record), QueuedTime (the time a Wrapper is submitted for this job), StartTime (the time the Wrapper starts executing the job), EndTime (the time the Wrapper finishes job execution and status update), and Status (submitted, queued, running, failed, succeeded). Each job should contain an executable command. In the case when the Network File System (NFS) is used (which is the common case in clusters among compute nodes), job commands should contain the necessary file paths needed by the job. When NFS is not used, job commands should contain input file paths relative to the execution directory on the compute node and in the meantime specify the corresponding absolute file paths on the machine where the files are stored (normally in the machine from which job submissions are made) using the –NonNFSFiles option through the Client so that those files will be staged into HDFS by the Client and later retrieved by the Wrapper and put into the correct directory structure relative to the command execution directory on the cloud node where the job is executed (in this case, there is a single globally accessible HDFS system and each cloud node also has its own local file system). Other supporting non-NFS input files can also be staged in this way. Output result files are however not staged automatically in the current CloudBATCH prototype. Users should include a file staging logic if necessary at the end of their programs to stage result files explicitly to the desired destinations. Please pay attention to the way how some of the columns are named and data are stored. For example, instead of having a single "Status" column, we use columns named by the corresponding status such as "Status:queued". This is designed for doing fast queries such as "all the jobs that are queued" by taking advantage of some methods provided by HBase that can rapidly scan for all the rows for columns with non-empty values.

The Scheduled Time Table (Table V) is used to record job requests with a specified earliest start time. Scheduled job information is first entered into the Scheduled Time table and then put into the Job table with "Status:submitted". The value of "ScheduledTime" in the Scheduled Time Table is used to set the "SubmitTime" in the Job Table. When a Job Broker sees a scheduled job, it will not process it until the "SubmitTime" has been reached.

The User Table (Table VI) is used to record some user-specific policies or restrictions such as the maximum number of simultaneous jobs allowed to be running in queues at any time. The "IndividualJobPriorityLimit" regulates the highest job priority value a user can submit a job with. This ensures that only authorized users can submit high priority jobs that may preempt lower priority jobs if desired (future work).

In the following subsections, we introduce the key CloudBATCH components one by one and discuss how well CloudBATCH satisfies the desired properties mentioned in Section III as well as several other design issues.

B. Client

The Client provides user interaction functionality for user access control and job submission to proper queues with the capability of staging Non-NFS files into Hadoop HDFS. The Client normally runs on the job submitting machine. Figure 3 shows the process of handling job submission.

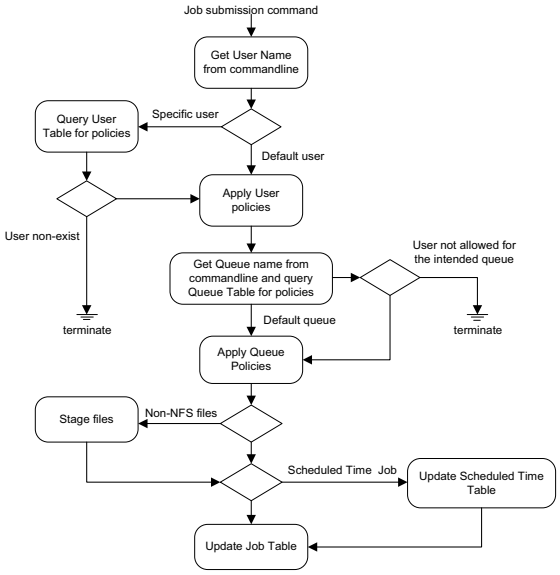

Figure 3. The flow of operations in the CloudBATCH Client.

After a job is submitted through the commandline interface, the Client parses the command for user and queue names (if unspecified, the default user and queue will be used). The User Table will be checked for access control purposes, and user policies such as the maximum job priority value allowed will be applied to the submitted job. The Queue Table will also be checked to see if the user is allowed to submit jobs to the specific queue and to apply queue policies to the job as well. If the policy from the Queue Table conflicts with the one from the User Table, an exception will be raised and the job should be submitted again with corrected information. After these checks, the Client finally gets a unique job ID by the mechanism of global sequence number assignment mentioned in [6] and puts the job information into the Job Table with "Status:submitted". If the job uses non-NFS files, there is currently some limited support from the Client to let users specify colon-separated pairs of

Table I
QUEUE TABLE

QueueID	QueueType	Capacity	JobPriority	JobLengthLimit	Domain	UserAlice	UserBob	Groupdefault
serial	serial	50	1	1200000	public			Y
bioinformatics	MapReduce	10		600000	private		Y	

Table II
JOB TABLE, PART 1

JobID	QueueID: serial	QueueID: bioinformatics	JobType:serial	JobType:MapReduce	JobType:ScheduledTime	Priority:1	Priority:5
23	Y		Y			Y	
24		Y		Y		Y	
25	Y		Y		Y		Y

Table III
JOB TABLE, PART 2

JobID	StartTime	EndTime	SubmitTime	QueuedTime	Job Length Limit	Status: submitted	Status: queued	Status: running	Status: failed	Status: succeeded
23			t1		1200000	Y				
24	t4		t2	t3	600000			Y		
25			t5		1200000	Y				

Table IV
JOB TABLE, PART 3

JobID	UserAlice	UserBob	Groupdefault	Groupbio	Command	JobDescription	AbsolutePath1	AbsolutePath2
23	Y		Y		Command1	mpeg encoder	RelativeFileDir1	
24		Y		Y	Command2	bio sequence		RelativeFileDir2
25		Y		Y	Command3	space weather		

absolute local file paths on the client machine and relative remote file paths on the Hadoop execution node (used in the job command) through the –NonNFSFiles option. The Client creates a column in the Job table (Table IV) named after the absolute local file path with column value the remote file path relative to the job command execution directory. The absolute file path, concatenated with a HDFS path prefix containing the job ID, will be used as the temporary file path in HDFS where the Client will transfer the user-specified local files to. For example, for job ID X with command "ls images/medical.jpg", the user can specify a file pair "/home/user/images/medical.jpg:images/medical.jpg". Then the local image file will be transferred to HDFS under "/HDFS/tmp/jobX/home/user/images/medical.jpg". The Wrapper that executes this job later will copy that HDFS file to the local relative path "images/medical.jpg" before invoking the job command. The executable of the command should be transferred this way as well if necessary. Future work can be added to distinguish between input files and output files so that non-NFS file users can have an option to put the result files back into HDFS with customized locations.

C. Job Broker

The Job Broker is responsible for polling the Job Table for lists of user submitted jobs that are ready for execution. For every job on the lists, the Job Broker submits to the Hadoop MapReduce framework a "Wrapper" for job execution. Figure 4 shows the tasks a Job Broker should perform periodically when polling the Job Table.

A Job Broker queries the Job Table periodically to get a list of jobs with "Status:submitted" according to job priorities from high to low. For each of the jobs in the list, the Job Broker checks whether the job is a scheduled job that has not yet reached its scheduled execution time, whether the intended job queue is already full to its maximum capacity, and whether the user who submits the job already has a number of running jobs at the maximum value of his allowed quota. If the job is allowed to be executed immediately, the Job Broker changes the job status to "Status:queued" and submits a Wrapper to execute it. Note that the job status is changed before submitting the Wrapper. If the Wrapper fails

Table V
SCHEDULED TIME TABLE

ScheduledTimeID	UserBob	Groupbio	JobID	ScheduledTime	RequestSubmitTime
2	Y	Y	25	t5	t0

Table VI
USER TABLE

UserID	SimultaneousJobNumLimit	IndividualJobPriorityLimit	Groupdefault	Groupbio	Groupspace
UserAlice	20	5	Y		
UserBob	30	8	Y	Y	

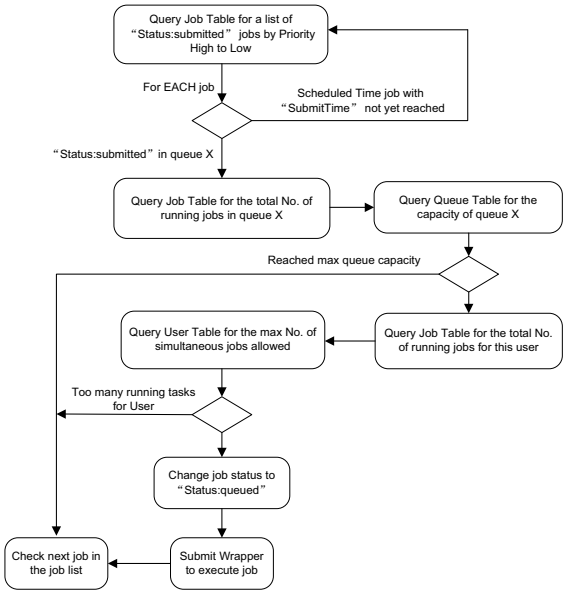

Figure 4. The flow of operations in the CloudBATCH Job Broker.

before actually starting the job execution, the job could be straggling with the "queued" status forever. This is handled by Monitors as described below.

D. Wrapper

Wrappers are responsible for executing jobs through the Hadoop MapReduce framework. They are actually Map-only MapReduce programs acting as agents to execute programs at compute nodes where they are scheduled to run by the underlying Hadoop schedulers transparently. When a Wrapper starts at some node, it first grabs from the HBase table the necessary job information and files that need to be staged to the local machine. Then it starts the job execution through commandline invocation, MapReduce and legacy job alike, and updates the job status to "Status:running". During job execution, it will check the execution status such as total running time in order to follow some job policies such as maximum execution time limit, and may terminate a running job if it takes too long. In the future, Wrappers can be extended to do more advanced resource utilization monitoring during job execution. After job execution completes, either successful or failed, the Wrapper will update the job status ("Status:successful" or "Status:failed") in the Job Table, perform cleanup of the temporary job execution directory and terminate itself normally.

E. Monitor

Monitors are currently responsible for detecting and handling Wrapper failures which may happen after Wrappers are submitted to Hadoop MapReduce but fail to execute the assigned jobs, which results in jobs being marked as "Status:queued" but never executed. The current method to achieve this is to first set a time threshold T, then let the Monitor poll the Job Table for jobs that stay in the "queued" status for a time period longer than T counting from "QueuedTime". It is possible that jobs will still be queued after a healthy Wrapper is submitted with it, even though queue capacity or user active job number limit are not reached. This may happen when the cluster is overloaded and newly submitted MapReduce jobs (i.e., Wrappers themselves) are queued by the Hadoop MapReduce framework. Therefore, the threshold shall be chosen and adjusted very carefully according to cluster size and load. Monitor usages are still not fully explored for our prototype and may be extended in the future for detecting jobs that have been marked as "Status:running" but actually failed.

F. Discussion

CloudBATCH supports the majority of the desired properties explained in Section III. For example, it adopts a scalable architecture where there is no single point of failure in its components (HBase is assumed to be scalable and robust) and multiple numbers of its components (such as Job Brokers) can be run according to the scale of the cluster and job requests it manages; it supports advanced user and job queue management with individually associated policies concerning user access control and maximum number of active jobs allowed, queue capacity, job priority, etc. It also inherits Hadoop's transparent data and computation deployment as well as task level fault tolerance concerning the way how Wrappers are handled. Some other properties are also supported by working complementarily with Hadoop schedulers, such as the minimum number of node

allocation guaranteed to each queue which can be specified inside Hadoop schedulers. In addition, CloudBATCH can handle simple non-NFS file staging requests rather than requiring users' manual efforts. The HBase table records can also serve as a good basis for future data provenance support. However, MPI is not yet supported since there is no Hadoop version of the mpirun command yet. The direct reservation of compute nodes rather than earliest start time specification for jobs is not supported either because Hadoop hides compute node details from explicit user control, a tradeoff for the purpose of transparent user access and ease of management over large numbers of machines.

Concerning fault tolerance, CloudBATCH is capable of handling two types of failures. The first is fault tolerance for jobs against machine failures. This is handled by the fault tolerance mechanism of the underlying Hadoop MapReduce framework through Wrappers. Because Wrappers are MapReduce programs, they will be restarted automatically by default for 6 times if machine failures occur. The other type of fault would potentially occur due to the concurrent execution of multiple CloudBATCH components. For example, if several Job Brokers run simultaneously, they might all try to update the status of the same job. This problem may be avoided by fail-safe component protocols which may possibly make use of distributed transactions with global snapshot isolation on HBase tables [6] so that updates to the HBase tables by different components won't conflict with each other.

V. Performance Evaluation

We present the result of a basic test for the server throughput of handling job submissions vs the number of concurrent Clients. The test is performed because according to the architecture of CloudBATCH, client job submission could become a performance bottleneck. The test investigates how fast the system can accept job submissions (inserting job information into the HBase tables with proper job status) from more and more concurrent users. The result provides insight in the system capacity of handling user requests. We assume that there is only one queue with no restrictions in terms of queue/user policies. Having multiple queues with complicated policies may decrease system performance, which will be studied in future work. The test is performed on a 3-machine cluster with Hadoop and HBase installed.

In the test, we run an increasing number of Clients with an even distribution of the number of Clients on each of our 3 machines concurrently. Each Client submits 100 jobs consecutively to our system. The result (Figure 5) shows that the system can accept about 25 job submission requests per second. We argue that the performance is satisfactory for normal user job submission request loads. The reason for the slight decline in throughput after reaching its peak value is because the Client components compete for the same system resources used to host Hadoop and HBase.

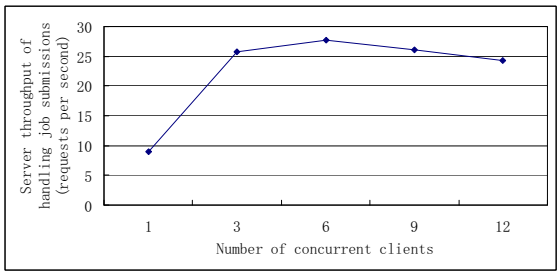

Figure 5. Server throughput of accepting job submissions vs Number of concurrent Clients.

VI. Conclusions and Future Work

In this paper, we describe a system called "CloudBATCH" to enable Hadoop to function as a traditional batch job queuing system with enhanced management functionalities for cluster resource management. With CloudBATCH, clusters can be managed using Hadoop alone to cater for hybrid computing needs involving both MapReduce and legacy applications. Future work will be explored in the direction of further testing the system under multi-queue, multi-user situations with heavy load and refining the prototype implementation of the system for trial production deployment in solving real-world use cases. CloudBATCH may also be exploited to make dedicated Hadoop clusters (when they coexist with other traditional clusters) useful for the load balancing of legacy batch job submissions.

References

[1] J. Dean and S. Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51:107–113, 2008.

[2] Hadoop. http://hadoop.apache.org/.

[3] HBase. http://hadoop.apache.org/hbase/.

[4] T. Sandholm and K. Lai. Dynamic Proportional Share Scheduling in Hadoop. In *LNCS: Proceedings of the 15th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 110–131, 2010.

[5] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 265–278, 2010.

[6] C. Zhang and H. De Sterck. Supporting Multi-row Distributed Transactions with Global Snapshot Isolation Using Bare-bones HBase. In *Proceedings of the 11th International Conference on Grid Computing (Grid)*, 2010.

[7] C. Zhang, H. De Sterck, A. Aboulnaga, H. Djambazian, and R. Sladek. Case Study of Scientific Data Processing on a Cloud Using Hadoop. In *LNCS: Proceedings of the 23rd International Symposium of High Performance Computing Systems and Applications (HPCS)*, pages 400–415, 2009.