Oozie: Towards a Scalable Workflow Management System for Hadoop

Mohammad Islam, Angelo K. Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann

Yahoo! Inc., Sunnyvale, CA 94089, USA

{kamrul, angeloh, mhb, mchiang, sms, cpeters, anew}@yahoo-inc.com

Alejandro Abdelnur

Cloudera Inc, Palo Alto, CA 94306, USA

tucu@cloudera.com

ABSTRACT

Hadoop is a massively scalable parallel computation platform capable of running hundreds of jobs concurrently, and many thousands of jobs per day. Managing all these computations demands for a workflow and scheduling system. In this paper, we identify four indispensable qualities that a Hadoop workflow management system must fulfill namely Scalability, Security, Multi-tenancy, and Operability. We find that conventional workflow management tools lack at least one of these qualities, and therefore present Apache Oozie, a workflow management system specialized for Hadoop. We discuss the architecture of Oozie, share our production experience over the last few years at Yahoo, and evaluate Oozie's scalability and performance.

Categories and Subject Descriptors

D.4.1 Processing Management: Scheduling

Keywords

Oozie, Hadoop, Workflow Management, Scalability

1. INTRODUCTION

Apache Hadoop [1] provides cloud-based services for batch-oriented data processing that have grown rapidly over the past six years. The challenge of creating a manageable and cost effective application execution environment for Hadoop has increased with this growth. During this period, many organizations have made significant investments in processing, data management, and human resources. For example, Yahoo currently operates over 40,000 compute nodes, processes over 100 Petabytes of data, and is used by hundreds of data scientists and programmers.

The bulk of the Hadoop workload consists of sequences of jobs written using a variety of tools and languages. Workflow provides a declarative framework for effective management of these diverse jobs. We considered several workflow implementations and found each of them lacking in at least one of the four major requirements: scale, multi-tenancy, Hadoop security, and operability. Based upon the identified gaps, we developed a new workflow system for Hadoop called Oozie.

Oozie [2] is architected from the ground up for large-scale Hadoop workflow. Oozie scales to meet the demand, provides a multi-tenant service, is secure to protect data and processing, and can be operated cost effectively. As demand for workflow and the sophistication of applications increase, Oozie must continue to mature in these areas.

SWEET 2012, May 20 2012, Scottsdale, AZ, USA

Scalability in Oozie takes a balanced approach combining both horizontal and vertical scalability characteristics. The balance is intended to allow organizations to choose whether to add more resources to existing servers or to add more servers to expand the capacity of the Oozie service as demand grows. The ability to effectively scale the Oozie service horizontally is an area of active research and development and explored in this paper.

Oozie is architected to support multi-tenancy. The shared service deployment of secured Hadoop allows organizations to optimize capital investment while maintaining the integrity of data and processing. The Oozie service isolates processing to assure that each user has access only to authorized resources. Oozie natively supports Hadoop security by executing workflows as the user in order to meet strict data security requirements.

Oozie provides organizations with visibility into the operational characteristics of workloads on Hadoop. A rich set of monitoring interfaces provides both a user interface for interactive management and APIs for integration with tools. This open architecture enables the cost effective operation of large workloads on Hadoop. Organizational needs for integration of operational metrics into applications that span platforms are growing; leading to new research into exposure of Oozie metrics in new ways.

To illustrate the current state of Oozie, we present some example statistics for production use at Yahoo. For the month ending 15 February 2012, Oozie managed over 770,000 workflows executing nearly 2 Million Hadoop jobs on 16 different Hadoop clusters. Each day during that period the most heavily used Oozie server launched over 13,000 Hadoop jobs for 50 different user applications. There were no outages or significant processing delays showing that Oozie is proven in the highly demanding Yahoo production environment. In order to maximize the value of investment in Oozie, it was contributed to the open source community in 2010. As more organizations adopt Oozie, innovation has accelerated through community contributions.

This paper is organized as follows: in section 2 we explore related workflow systems and identify their shortcomings with respect to Hadoop requirements. Section 3 describes in detail the architectural attributes of Oozie that enable scalability, multi-tenancy, and effective operability. In section 4 an experimental setup for measurement and the production setup at Yahoo! are described. Section 5 discusses using the experimental setup to measure the ceiling for several of the scaling limiters and discuss how the architecture behaves in Yahoo! production clusters. We describe some areas for ongoing research for Oozie in section 6. And finally we summarize our ongoing work in section 7.

2. RELATED WORK

Workflow management is a mature field addressed by both commercial vendors and academes. However, we find that there is no full-fledged workflow management system appropriate for Hadoop. In this section, we discuss a few existing workflow management systems along with their benefits and limitations with respect to Hadoop.

Commercial vendors such as Oracle, IBM, Pegasystems, EMC, and more have developed commercial workflow systems. None has yet created a way to configure Hadoop jobs within their workflow applications making any integration very loosely coupled at best. The lack of support of Hadoop security by commercial vendors could not be overcome easily. Oracle provides a Big Data Appliance [3] that includes Oozie as a part of the Cloudera's Distribution including Hadoop [4]. IBM InfoSphere BigInsights [5] includes a scheduler, but not a workflow system.

The e-Science [6] project intends to define scientific workflows for processing large distributed data, which is usually owned by different organizations. The project also established a framework for researchers to define and reuse workflows. Kepler [7], an e-Science tool, is mainly used to create workflows and share data models. The workflows are defined using the Kepler graphical interface and data is stored either locally or remotely. Taverna [8] is another e-Science tool to construct and execute workflows using local and/or remote data. Although the e-Science tools help gather data and models from different parties and encourage scientific collaboration between organizations, the ability to process large amount of data and placement of task executions on a distributed environment is still very limited. When the amount of data increases exponentially in today's scientific data processing, a workflow system that works with Hadoop can bring many benefits. MapReduce on Hadoop, which is built from commodity hardware, provides a parallel and scalable model for scientific analysis. While there is research on eScience tools [9] to integrate with Hadoop, it is mainly to create specific action node to execute MapReduce logic within a workflow and not to place the workflow execution on the Hadoop MapReduce framework. Without the delegation of workflow execution to a distributed framework like Hadoop, a workflow system does not take advantage of parallel task execution when the system resources can be re-assigned to other tasks. Thus the horizontal scalability of the e-Science workflow system will remain constrained.

One of the commonly used workflow management systems for various scientific applications is Pegasus. The Pegasus [10] workflow management system provides a comprehensive solution for constructing and enacting scientific workflows. Pegasus enables the workflows to be executed locally, on the grid, and/or on the cloud in a simultaneous manner. The system provides a rich set of APIs to construct the workflows as a Directed Acyclic Graph (DAG). The DAG, which is represented in an XML format, provides detailed information about the tasks, the input data set and the generated data set. Based on the abstract DAG, Pegasus constructs an executable workflow by querying a catalog of the existing set of local and distributed computing resources. Pegasus offers a comprehensive set of interfaces to various job scheduling systems to help execute and monitor the execution of the workflows. During the workflow execution, Pegasus monitors the execution performance and the reliability metrics, and thereby enables mechanisms to implement several failover and optimization techniques. Several publications [11][12] have proposed optimization algorithms to address the Pegasus workflow compilation and execution overheads.

Azkaban [13] is a batch job scheduler that can be considered as an amalgamation of the *UNIX cron* and *make* utilities. The system can execute jobs periodically where each job (a.k.a. workflow) could be a list of interdependent processes. It uses a series of properties files to specify a workflow and includes a graphical user interface to track the progress of jobs. Azkaban does not implement a security model, and thus support for multi-tenancy is limited. Jobs execute as the administrative user and not as the application user who submits the job. Azkaban does not support authorization for its job submission and control. Azkaban does not meet the multi-tenancy requirements of Yahoo due to the lack of process or data protection. The states of all running workflows are managed in memory. Upon system failure or restart all running workflows' states are lost. Azkaban executes all jobs as part of a single server process impacting scalability and increasing operational complexity. Any user application could negatively impact the server process. The execution path of the workflow is evaluated at submission time limiting the workflow semantics supported. Yahoo has many use cases in which control flow needs to be determined dynamically at runtime. Azkaban is very easy to use for simple applications; however it lacks significant features required for Yahoo.

Pig [14][15] and Hive [16][17], an imperative and a declarative language respectively, offer programming paradigms that resolve the language primitives into a series of MapReduce jobs. Pig and Hive create an execution plan for each script based on data and the processing logic. These programming paradigms are well suited for simple data processing applications. However, they are not suitable for running any type of user-defined jobs that require flexible flow control. Furthermore, Pig and Hive lack operational and monitoring support for life-cycle management such as suspend/pause/resume/kill of each job. While Pig and Hive internally execute a set of auto-generated MapReduce jobs in sequence, they do not support the Yahoo! workflow system requirements.

3. OOZIE FEATURES AND FUNCTIONS

Oozie is designed as a Java based Web application. In addition to supporting workflow management service, Oozie attempts to address a lot of common issues for any multi-tenant service. In this section, we first describe key constituents of Oozie's internal that plays important role in overall performance. Then we discuss how Oozie addresses the scalability, multi-tenancy, security and operability in subsequent sub-sections.

3.1 Oozie Internals

Figure 3-1 shows the key components of Oozie internal architecture. As shown in the figure, Oozie server provides a REST API support where each request is being authenticated by pluggable authentication module. After the workflow submission to Oozie, workflow engine layer drives the execution and associated transitions. The workflow engine accomplishes these through a set of pre-defined internal sub-tasks called *Commands*. Most of the commands are stored in an internal priority queue from where a pool of worker threads picks up and executes those commands. There are two types of commands: some are executed when the user submits the request and others are executed asynchronously. Oozie has a built-in auto-recovery mechanism where a dedicated thread periodically runs to monitor the progress of all the active jobs and take necessary action if any job stuck in some state. Additionally, there are two other daemon threads: one

for purging the old records from DB and other one to check the external status of any submitted Hadoop job. Oozie supports multiple databases such as Oracle, MySQL, Apache Derby, etc. to store the internal states through a generic persistence layer. The use of a relational database provides the benefits of transactional state management. For instrumentation purpose, Oozie provides a shared layer to make the statistics available to the administrator. Moreover, Oozie provides a generic Hadoop access layer restricted through Kerberos authentication to access Hadoop's Job Tracker and Name Node components.

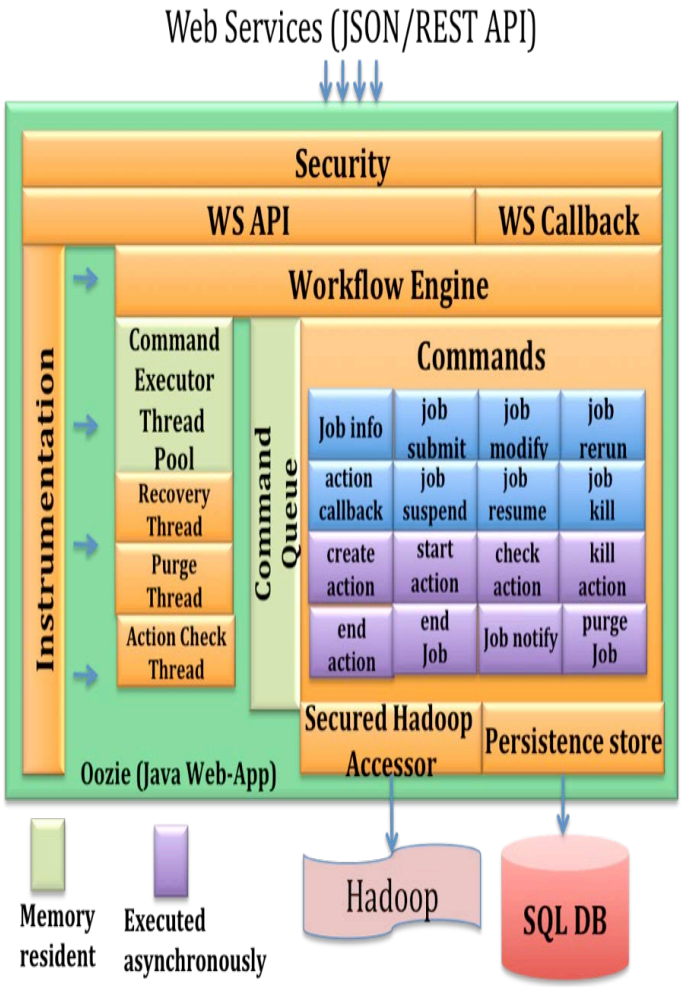

Figure 3-1: Oozie internal components

3.2 Scalability

Scalability is an essential feature for any workflow management service. In cloud-based systems, load increases are handled by design. In general, there are two types of scalability: horizontal and vertical. Horizontal scalability is the ability for multiple servers to concurrently act as a single logical service thus new servers can be added to increase capacity. Vertical scalability refers to increasing capacity by adding extra resources to the same machines. In this section, we describe the challenges that we faced in achieving horizontal and vertical scalability, solutions to some of these challenges along with a note on ongoing work.

3.2.1 Horizontal Scalability in Oozie

Oozie supports horizontal scalability by design. There are two major design decisions that play an important role in this context. Firstly, Oozie needs to execute different types of jobs as part of workflow processing. If the jobs are executed in the context of the server process, there will be two issues: 1) fewer jobs could run simultaneously due to limited resources in a server process causing significant penalty in scalability and 2) the user application could directly impact the Oozie server performance. Therefore, in Oozie, we opted to execute actual jobs on launcher machines that are managed by Hadoop. Subsequently, if the number of concurrent jobs increases, Oozie can use more compute nodes in a cluster making it horizontally scalable.

Secondly, Oozie stores the job states into a persistent store. This approach enables multiple Oozie servers to run simultaneously from different machines. Multiple servers could distribute the tasks among themselves. The coordination of these multiple servers to distribute the task load is another critical concern. Zookeeper [18], a distributed coordination system, could be used in this regard. Although a Zookeeper service can be plugged into the system as part of the current design, it's an area for future work. Note that the use of a relational database to manage state for *infinitely* scalable system design is limited by technical and cost considerations. There is an on-going discussion to utilize other low-cost, efficient and highly scalable persistent mechanism (such as Zookeeper) instead of a relational database.

3.2.2 Vertical Scalability in Oozie

Horizontal scalability is an integral attribute of Cloud computing where machines are virtually infinite and easy to deploy - provided the application is designed accordingly. On the other hand, vertical scalability is not directly related to the Cloud. It primarily depends on the application design and development. In general, vertical scalability is achieved by adding resources (such as memory) to an existing service to increase capacity.

In an effort to achieve vertical scalability, we followed some important design principles to reduce the impact on memory or I/O burden. First, as described in the section on horizontal scalability, we chose to store the job state into persistent storage instead of retaining it in memory. In addition, large tasks are broken down into smaller sub-tasks that could be executed within a short period of time (usually few milliseconds) and persist the intermediate states upon the completion of the sub-task. To reduce I/O overhead, minimal information is persisted for state transitions. When load increases there is no extra consumption of memory because no job information is in memory. Although not directly related to scalability, it provides a highly desirable side-effect: fault-tolerance; if the service is restarted, it can resume any job from the persisted intermediate state without any user intervention. These design choices lay the groundwork for vertical scalability at the expense of slight performance degradation in some instances due to frequent updates to persistent storage especially with high I/O traffic and overhead. Moreover, as noted in the previous sub-section, the use of a database as persistent storage has shortcomings to achieve *true* scalability.

As mentioned above, Oozie splits larger workflow management tasks (not Hadoop jobs) into smaller manageable subtasks and *asynchronously* processes them using a pre-defined state transition model. Every state transition is executed through a set of subtasks called `Commands`. Commands are stored in a fixed size, in-memory command queue. A pool of worker threads executes these commands. The size of the command queue and the size of worker thread pool are the key components that have direct memory impact. Both the queue size and thread pool size are pre-defined and configurable. The correct value for these settings is determined based on available memory and expected

load. If the load increases beyond the configured limits, Oozie operates without failures with slightly degraded performance. The ideal solution in these situations is to increase the memory and consequently the Oozie service capacity will be increased.

Oozie is designed to minimize the impact of failures in external systems, such as Hadoop (Job Tracker and Name Node) and the persistent store. Although it is primarily related to fault-tolerant features, it could directly impact the vertical scalability if the resources are unnecessarily consumed beyond the required duration due to these external failures. Therefore, Oozie often uses a pre-defined timeout for any external communication. Oozie follows an asynchronous job execution pattern for interaction with external systems. For example, when a job is submitted to the Hadoop Job Tracker, Oozie does not wait for the job to finish since it may take a long time. Instead Oozie quickly returns the worker thread back to the thread pool and later checks for job completion in a separate interaction using a different thread. In order to maximize resource usage, the persistent store connections are held for the shortest possible duration. To this end, we chose a memory lock based transaction model instead of a persistent store based one; the latter is often more expensive to hold for long time. All of these techniques make sure that Oozie resources are utilized efficiently, which directly improves the vertical scalability.

In summary, Oozie accepts this *diagonal* approach by incorporating the best solution present in both vertical and horizontal scaling for a workflow system. While Oozie demonstrates scalability in production at Yahoo!, opportunities for improvement remain.

3.3 Multi-tenancy

Multi-tenancy [19][20] is the name given to a software architecture in which one instance of an application, running on a remote server, serves many client organizations (also known as *tenants*). Multi-tenancy is the core tenet of cloud computing.

Oozie separates tenants in two ways. Firstly, while Oozie utilizes a shared database instance for all tenants, each tenant's data is segregated by an Oozie designated identifier. When a user submits a workflow job, Oozie first authenticates and later validates the user's privileges with the underlying security implementation. The user has the privileges to create, read, update, delete and reprocess jobs. All workflows are saved in a shared database schema whose access is controlled. As no new database or software resources need to be created for each incremental tenant, the cost of on-boarding a new tenant approaches to zero.

Oozie is also a tenant to Hadoop. Multiple Oozie instances that run on separate databases can be installed and configured to run jobs on a single Hadoop cluster.

The integrity of Oozie tenants' data is never compromised. A workflow job can execute different kinds of tasks such as MapReduce, Pig, Java, etc. The data or script of a job is not saved in Oozie instance, but underlying cluster storage, which is secured and accessed only by the job owner. Although Oozie users share system resources and storage for workflow execution and job definitions respectively, the processing power and storage of tasks are segregated by each user and configured independently.

Oozie provides a flexible mechanism to maintain and control the environment of an instance. The system resources and database utilization can be controlled per Oozie instance. In a highly loaded system, the memory resources and thread concurrency can be tuned with suitable values. Oozie provides system configurations from higher level, such as maximum number of same type of tasks processing by system worker threads, to lower level, such as number of worker threads in a pool.

Oozie provides workflow software as a service. Traditionally, a user purchases a workflow software package and license by paying a one-time fee. Creation and maintenance of workflows in a parallel execution environment like that of Hadoop can be very complex. Security is also an essential component for executing jobs on Hadoop system. Workflow as a service reduces the need for a tenant to build a redundant security layer and a workflow monitor. Existing Hadoop users can easily migrate to the Oozie workflow service and create Oozie workflow with pre-defined tasks.

3.4 Security

Cloud security [21][22][23] primarily focuses on two areas: 1) application security, and 2) data security and privacy. In addition, it is desirable to provide a pluggable security framework where any organization-specific security or identity mechanism can easily be included.

Firstly, Oozie ensures that its service is secure, thereby satisfying the Cloud based application security criterion. The security model includes enforcing authentication and authorization for every incoming request to the Oozie service. For example, Oozie should make sure that a user should not kill a workflow submitted by another user without having the right privileges. Oozie, by default, supports the Kerberos based authentication and Unix based user/group authorization mechanisms. In addition, Oozie offers a pluggable security module allowing any business unit to use other off the shelf or proprietary authentication mechanisms. Oozie also supports multiple authentication mechanisms wherein different applications can submit requests using different credentials to the same Oozie service. For example, in production at Yahoo!, Oozie simultaneously supports two different proprietary authentication mechanisms in addition to the Kerberos based authentication mechanism. Furthermore, the security model provides a way to impersonate any pre-configured super-user in Oozie to submit requests on behalf of another user to Oozie. This requirement is essential for any multi-tenant service built on top of the Oozie service.

Secondly, while Oozie depends on Hadoop to enforce data security built on Kerberos [24][25] Oozie has to make sure that the *appropriate* Kerberos token is available for every application submitted from Oozie to Hadoop. In particular, it is more challenging to make the Kerberos token available in a multi-tenant setup. While the multi-tenant service often runs as one user, say the administrative user, Oozie executes those as different users; say the application users who submit the job. In general, Oozie asynchronously submits jobs to Hadoop. As a result any application user's credentials could have expired at job execution time which could occurs long after job submission In summary, the job executor is the administrative user whereas the real job execution and data access need to be carried out as the application user. Oozie addresses this requirement using a Hadoop feature where the administrative user is pre-configured as a super user in Hadoop allowing Oozie to impersonate [26] the application user. In this case, Hadoop only checks the credentials from the administrative user assuming that Oozie has already authenticated the application user during job acceptance and subsequently executes the job in Hadoop as the application user.

3.5 Operability, Reliability and Monitoring

In large-scale multi-tenant computing environments workflow services must operate with minimal manual intervention reliably

and expose the ability to manage the workflows to clients of the grid.

Oozie supports a large degree of deployment automation through externalization of configuration. At Yahoo! the deployment of an upgrade to the Oozie service takes only a few minutes. Oozie generates extensive logs for the identification and diagnosis of operational issues.

Applications are being introduced and evolving at a rapid pace. Large-scale real-world data streams occasionally have errors. Thus any workflow service must be both reliable and allow for re-execution without the need to start from the beginning. Oozie provides a mechanism by which a failed workflow job can be resubmitted and executed starting after any action node that has completed its execution successfully in the prior run. This is especially useful when the already executed actions of a workflow job are too expensive to be re-executed. To enable such a behavior, Oozie enables its users to submit their workflow in a recovery mode. When starting a workflow job in recovery mode, the user must indicate either what nodes in the workflow should be skipped or whether the job should be restarted from the failed node. A recovery run can be executed using workflow job parameters that are different from that of the original run; the new values of the parameters will take effect only for the workflow nodes that are executed in the rerun. The workflow application use for a re-run must match the execution flow, node types, node names and node configuration for all executed nodes that will be skipped during recovery.

In addition, Oozie provides an automatic retry capability when an action encounters an error. The automatic retry capability allows users to give certain number of reties. If a failure or an error persists after max retries, the action remains in an error state and the system marks the workflow job appropriately. The application or user then has the option to resubmit the workflow from the start or execute from the location where the error occurred.

Oozie provides a failure management mechanism in which the system will be able to recover the status of all its running workflows in case of any unexpected failure. The Oozie recovery management module utilizes the persistence store to manage checkpoints of the workflow execution progress. In case of failure, the Oozie server will restart from its latest checkpoint. Although the current implementation doesn't enable automatic failover between the primary and secondary servers, the existing mechanism will make it easy for automatic failover, which is planned as part of our future work.

Oozie can detect the completion of a MapReduce job by two different mechanisms, callbacks and polling. When Oozie starts a MapReduce job, it provides a unique callback URL as part of the MapReduce job configuration; the Hadoop Job Tracker invokes the given URL to notify the completion of the job. For cases where the Job Tracker failed to invoke the callback URL for any reason (i.e. a transient network failure), the system has a mechanism to poll the Job Tracker for determining the completion of the MapReduce job. In case of a detected failure, Oozie will update the status of the job and its corresponding action and may trigger an automatic rerun if configured.

Oozie is internally instrumented to capture extensive operational information both about the executing workflows and about the operational characteristics of the service. A comprehensive set of APIs make this information available via command line, Java API, and a RESTful HTTP web service.

The comprehensive security, scalability, and operational functionality of Oozie make it the default choice within Yahoo! for workflows that run in the Hadoop environment.

4. EXPERIMENTAL SETUP

As part of the empirical analysis of Oozie's performance, we gathered results from two sources. First, we collected the data from various Yahoo! production clusters where multiple Oozie instances are running. Production data represents the realistic system usage and is very valuable. However, as part of a defensive strategy, most production systems are over-provisioned so that they rarely reach their maximum capacity to avoid downtime. Therefore, it is hard to determine the resilience of a system in any production environment. However, it is important to estimate the system scalability and performance metrics in a high stress environment. Hence, we artificially create a highly loaded environment to evaluate the key metrics.

4.1 Emulated Setup

In this case, we adjust our experimentations in three different layers. First, in the Oozie client that actually submits the requests to Oozie. The variation in the job mix of the submitted workflows is important to mention. Then at the Oozie server layer which ultimately executes the workflow. Lastly, in the Hadoop cluster where the job actually runs. In this section, we discuss the important setup issues for each layer.

In our experiment, we emulate the simultaneous job submission by multiple users through multiple client sessions. Subsequently each client uses multiple threads for job submissions. We use different number of jobs per workflow to evaluate the impact on performance metrics in varied combinations. Each client machine has a 2 x Xeon 2.50GHz (8 cores) CPU with 16GB RAM, running RHEL Server 5.6 64-bit OS. Even though the client box has 16 GB of memory, the client emulator only utilizes 1 GB memory in our experiments. The exact details of number of clients, number of threads per client and the number of jobs per workflow will be discussed in the relevant result sections.

The Oozie server machine has a 2 x Xeon 2.50GHz (8 cores) CPU with 16GB RAM, running RHEL Server 5.6 64-bit OS. Although the machine has 16 GB memory, the Oozie process is started with 3 GB RAM. In addition, the Oozie server is configured with 300 worker threads with an internal queue size of 10K. The experimental Hadoop cluster, as the last layer, is comprised of 7 nodes. The configuration of each node is similar to that of the Oozie server machine described above. The cluster is configured with a maximum 28 map slots, which means there could be at most 28 mapper applications running at any instance.

4.2 Production Setup

Yahoo has very large Hadoop installation composed of nearly 42k nodes that are divided into more manageable clusters for production, research and sandbox. There is one Oozie service running per Hadoop cluster. In this section, we present the data from a cluster that contains 4k nodes. The total number of users in this Oozie instance is about 50, where each user can be running multiple applications.

5. EXPERIMENTAL RESULTS AND ANALYSIS

In this section we explain the data collected from emulated and production setups. Emulated setup provides us more flexibility of evaluating the performance from multiple perspectives. On the

other hand, production data gives us the usage pattern and some realistic metrics.

5.1 Request Acceptance Rate

It is a desirable feature for any multi-tenant cloud service to simultaneously accept requests from multiple users. The number of such requests could be very large in a high load situation. We wanted to evaluate how Oozie performs in such a condition. For experimentation, we compute how many workflow submission requests Oozie can accommodate per minute. We choose workflow submission request over other request types for two reasons. First, submission request demands a lot of resource accesses. For example, Oozie reads the workflow definition from a file and parses it. After successful parsing, it stores the information into a persistence store and returns the workflow identifier to the client. Therefore, we consider that this type of request is more resource intensive than other request types and could demonstrate the insight better. Secondly, a user usually submits a lot of workflows all at once, which are scheduled to run much later over a period of time. This realistic use case also prompted us to choose submission request type as the basis for comparison.

We use our experimental setup, defined in section 4.1, for this study. We imitate workflow submission by multiple applications or users by running workflow submission using multiple client sessions. In turn, each session spawns multiple threads to submit workflows at the same time.

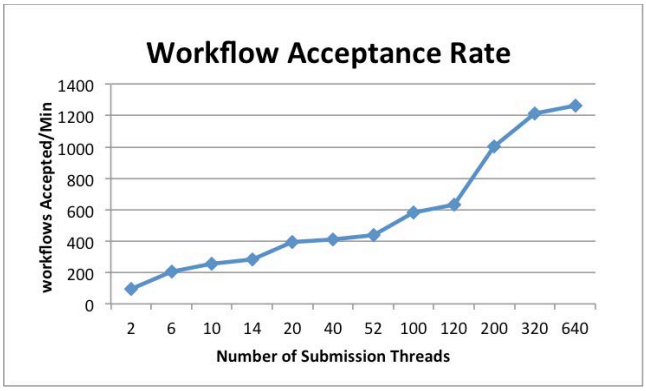

Figure 5-1: Number of workflows accepted by Oozie with incremental submission threads

Figure 5-1 shows the number of workflows accepted per minute by Oozie while the number of submission threads varies. We calculated the number of workflow submissions by multiplying the number of client sessions and number of threads per session and the number of submission per thread. For instance, we used 1000 workflow submissions per thread; that means we submitted as high as 640K workflows. The figure clearly reveals that the acceptance rate increases when the number of threads grows. It also demonstrates that Oozie can accept more than 1250 workflows per minute (75k/hour), a reasonably high value for one Oozie service instance.

5.2 Impact of Queue Size on Scalability

As described in section 3.2.2, vertical scalability can be accomplished by adding extra resources (such as memory, I/O bandwidth etc) to serve the increased load. Also mentioned above, Oozie uses two data structures of pre-defined size in memory: the internal queue and the pool of worker threads to process the queued task. In this sub-section, we strive to show how to improve the vertical scalability by adding extra memory and, subsequently bumping up the maximum queue length to process the extra load.

Figure 5-2 shows that the 52K workflows are submitted in a burst mode within the first two hours. It further displays the workflow load (number of active workflows) as the time progresses. The chart shows that most of the workflows are completed within first eight hours. Figure 5-3 presents the Oozie internal queue size for the same time duration as it processes all the workflows while the maximum queue length is fixed at 10K. According to this figure, the queue utilization reaches the maximum level within first half an hour and it stays at the same level for the next four hours. The rest of the time (11 AM – 3 PM) the queue size is nearly zero while there are a lot of active workflows in the system. The reason is that all the jobs for workflows are already submitted to Hadoop and Oozie is just waiting for those jobs to complete. In this situation, activities within Oozie are at a minimum and therefore queue size is close to zero.

In this experiment, we demonstrate that Oozie's workflow load (number of active workflows in the system) and internal queue length are directly related. As a result, if the load increase is anticipated, the system memory should be increased and, at the same time, the queue maximum length should be bumped up accordingly to accommodate the extra load. It is important to note that, in this experiment, we only consider the impact of queue size on vertical scalability assuming its other co-factors are already scalable. Work is planned to further investigate the impact considering the multiple factors such as CPU utilization, I/O and database connections simultaneously.

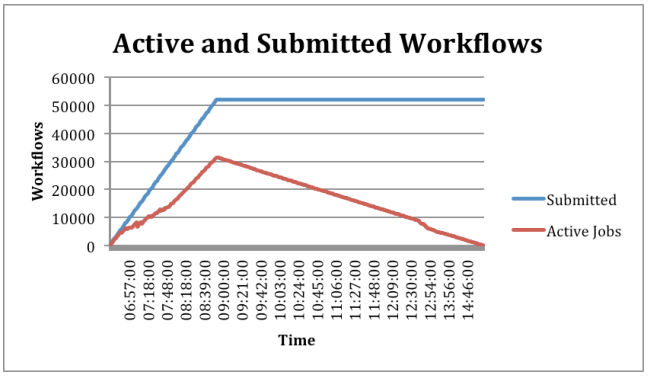

Figure 5-2: Number of workflows as the time proceeds

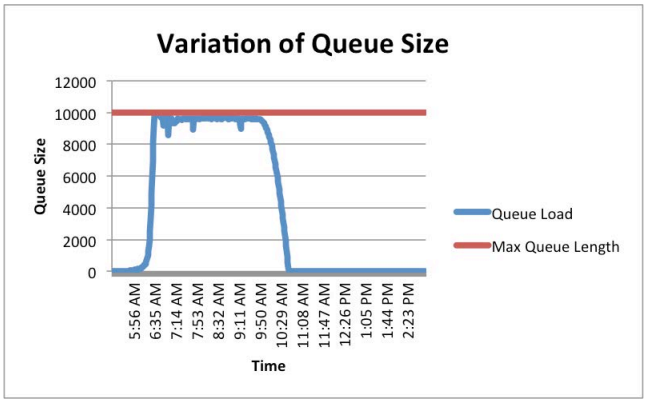

Figure 5-3: Variation of Queue size as the time proceeds

5.3 Overhead Analysis

One of the main goals of our experiment is to identify the overheads imposed by Oozie during the workflow execution. Our

experiments are tailored toward to dig deep to allow us to further explore optimization techniques to reduce these fixed costs. In this section, we will investigate the results of our experiments, and analyze the impact of these overheads.

For our analysis, we used execution logs gathered from Oozie and Hadoop. We identified the following measurement points to help us analyze the behavior and the fixed costs of the workflow execution:

Oozie Workflow Acceptance Time [T_{WA}]: The time at which Oozie accepts a user-defined workflow.

Oozie Action Start Time [T_{AS}]: The time at which Oozie starts realizing a workflow action and pushing it into its internal queue.

Hadoop Job Submission Time [T_{JS}]: The time at which Hadoop accepts the MapReduce job and pushed the job into its internal queue

Hadoop Job Finish Time [T_{JF}]: The time reported by Hadoop indicating that the MapReduce job has concluded

Oozie Action End Time [T_{AE}]: The time reported by Oozie indicating that the Oozie action has concluded

Oozie Workflow End Time [T_{WE}]: The time reported by Oozie indicating that the Oozie workflow has concluded

Based on the measurement points identified in the previous section, We we identified the following two types of overheads

$$O_{AP} = \left(T_{WE} - \left(\sum_{\forall Actions} \left(T_{AE} - T_{AS} \right) \right) - T_{WA} \right) / \, Number\_Of\_Actions$$

encounterd during the exection of workflows:

Action Preparation Overhead [O_{AP}]: The time consumed by Oozie to prepare the exection of its actions in adition to the time consumed while tarnsitioning from the action end to the start of a new action in the workflow. **O_{AP}** is normalized based on the number of actions in the workflow to get the per action overhead.

O_{AP} can be calucated as in the following formula:

Action Launching Overhead [O_{AL}]: The time required by Oozie to distribute the job metadata to the Hadoop cluster in prepration for launching the Hadoop job in addition to the time taken by Hadoop to inform Oozie about the conclusion of the job.

O_{AL} can be calculated as in the following formula:

$$O_{AL} = \left(T_{AE} - \left(T_{JE} - T_{JS} \right) - T_{AS} \right)$$

Figure 5-4 shows the phases of the execution of a single job; illustrating the six measurement points. As well, the dark gray circles indicate the action launching overhead (**O_{AL}**) and the light gray circles indicates the action preparation overhead (**O_{AP}**) for a single action. For multiple actions, the transition between the action end time and the start of a new action in the same workflow will be included in the calculation.

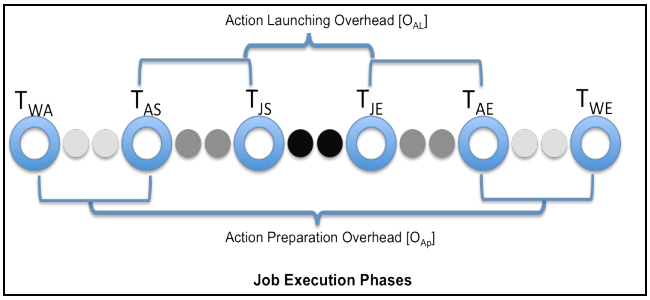

Figure 5-4: Illustration of the measurement points and the overhead

To better understand these overheads, its dependencies and distribution, we ran several experiments. Each experiment had a different number of actions per workflow, namely one, five, ten and fifty actions. To simplify our analysis, the action execution is serialized using a simple workflow execution graph. We are planning to consider more complex workflow executions graphs for futher reseach..

Figure 5-5 illustrates the total per action overhead (**O_{AP}+O_{Al}**) imposed by Oozie. We can clearly observe that more the number of actions per workflow, the less the overall per action overhead. In other words, the overhead required to prepare and launch the first action is greater than the overhead required to transition between actions.

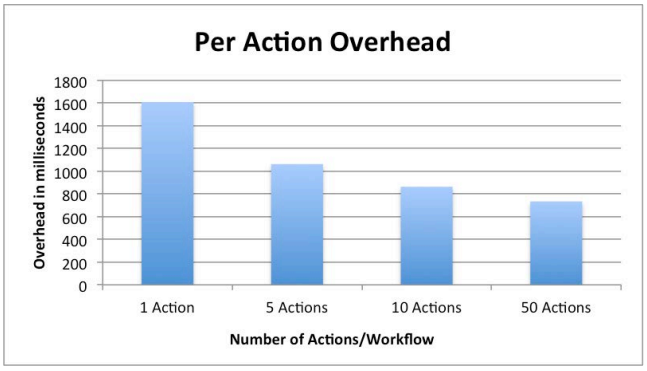

Figure 5-5: Per action overhead for different actions/workflow

Digging into more details about the types of overheads explained previously. Figure 5-6 and Figure 5-7 show the probability distribution of the action preparation overhead and the action launching overhead respectively; illustrating a reduction in the standard deviation of the overhead as the number of actions increases. In other words, the overhead becomes more deterministic with more actions per workflow.

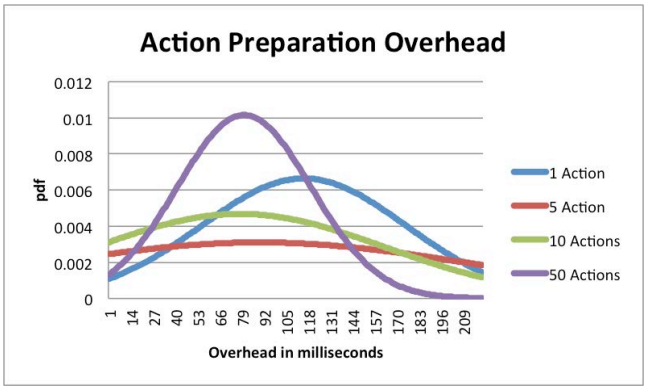

Figure 5-6: The distribution of the action preparation overhead for different actions/workflow

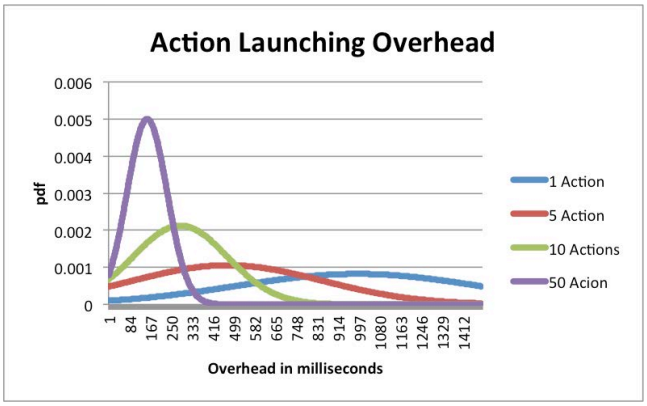

Figure 5-7: The distribution of the action launching overhead for different actions/workflow

5.4 Production Results

One of our intentions was to share our production experiences of serving a large system with Oozie at Yahoo. Consequently we present a set of graphs that show the usage pattern. Since we can't adjust any system knobs and usage pattern in a running system, the analysis and subsequent conclusion might be less substantive.

5.4.1 Distribution of Job Types at Yahoo!

Figure 5-8 shows the percentage of each type of Hadoop job that users submitted to two Oozie production servers over the period of a month. The figure reveals that nearly three fourths of jobs that are submitted to Oozie are Java processes and Pig scripts. An interesting observation is that users prefer the use of Pig over MapReduce by a wide margin. In addition, the overall distribution of job types is consistent among these two production clusters.

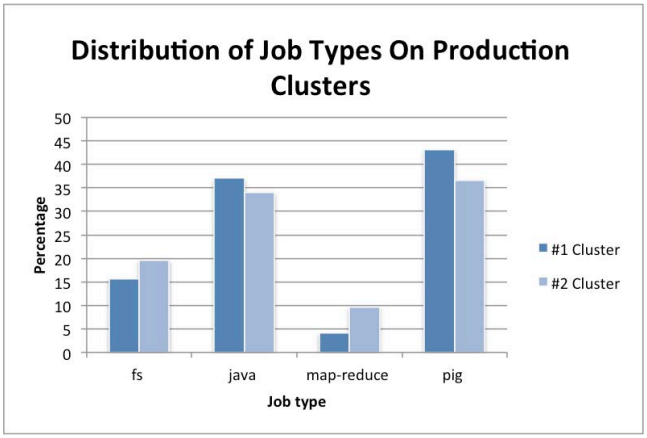

Figure 5-8: The distribution of the different type of jobs submitted by Oozie at two Yahoo! Production clusters

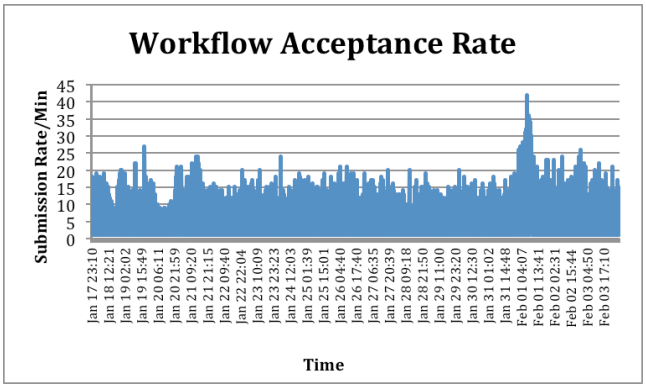

Figure 5-9: Number of workflows accepted by Oozie per minute

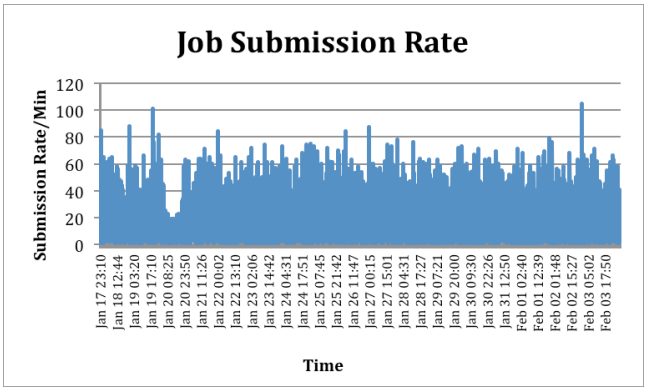

Figure 5-10: Number of jobs submitted by Oozie per minute

5.4.2 Typical Load at Yahoo

Figure 5-9 and Figure 5-10 show the rate of workflow acceptance and job submission respectively. It is mainly an illustration, on a per minute basis, of the number of workflows accepted by Oozie and the jobs submitted through Oozie during the period from Jan 18 to Feb 4 on our 4K nodes cluster. Around 40K workflows were accepted and 110K jobs were submitted during this period, with an average of 2.5K workflows accepted and 7K jobs submitted per day.

6. FUTURE WORK

Since its inception, Oozie has come a long way to provide an effective workflow management service for Hadoop-based data processing systems. Oozie's constant improvements are accomplished through the experiences gained in from companies such as Yahoo, and Cloudera, as well as the diverse feedback received from the Apache open source community. As data size is constantly increasing and the data processing use-cases are further diversifying, Oozie has to address a few important issues. In this section, we discuss the possible features for the coming releases.

As we discussed in section 3.2, Oozie needs to address some scalability issues. First, for horizontal scalability we have to implement load balancing in Oozie. Although the basic infrastructure is already there, we are considering utilizing Zookeeper for the coordination among multiple Oozie instances. In addition, we want to evaluate the impact of alternative technologies for persistent storage (such as Zookeeper) to replace the current SQL database that has known limitations in true scalability. To improve the vertical scalability, we intend to minimize the data size required to update the states, which is expected to reduce the I/O bandwidth substantially.

The experimentations mentioned in section 5 are the first attempt to quantify Oozie's performance along multiple dimensions. We found that some of our claims can not be strongly established without additional data and further analysis. Therefore we plan to expand the scope of our current investigation by adding more metrics and associated analysis. The experiment discussed in this paper was carried out against a small cluster of only 7 nodes. We need to use a larger cluster for our experiments to better represent production environments. In addition, our workflows were a linear DAG. We plan to profile the DAGs used in production applications and experiment with a representative mix. We expect the planned experimentations would assist us to identify the key factors of Oozie overhead and subsequently could lead to improved workflow processing performance.

In our current design, as soon as any Hadoop job is ready to be executed, Oozie submits it to Hadoop without considering the availability of Hadoop services. If Hadoop is highly overloaded or unavailable due to maintenance, the user needs to resubmit the job when the Hadoop server is back. This is clearly very inconvenient for the user. For example, a user who submits a workflow on Friday and expects the result to be ready when she is back on Monday morning will be hampered by unplanned Hadoop downtime. We plan to design Oozie to detect the Hadoop downtime and subsequently reduce the user's burden. In this case, Oozie could throttle the submission as soon as it identifies that Hadoop is down or slow. When it finds that Hadoop is back to normal, Oozie can submit those pending submissions.

Effective systems management requires the ability to monitor workflows. Oozie has a basic user interface built on Oozie web service APIs. We find that every business unit has different monitoring and alerting facilities which are difficult to meet with a uniform UI. Therefore, we plan to extend our web service monitoring API so that any business unit can build a customized monitoring/alerting system. Moreover, Oozie currently provides notification of workflow events (such as succeeds, killed or failed jobs) through web service callback and email. We intend to add additional flexibility through notification via a message bus.

Oozie has some usability concerns as well that we plan to address in phases. Defining a workflow in XML is verbose and not easy to write. We plan to simplify the XML definition through modularizing the different configurations and allowing reuse. In addition, we plan to explore the option of providing a modeling tool to easily create the XML and subsequent inclusion of the tool into an IDE. At the same time, we want to study other declarative and easy-to-use languages that can effectively define a workflow without XML.

7. CONCLUSION

Oozie represents a major advancement as a scalable, multi-tenant, secure, and operable workflow service for Hadoop. We have illustrated the need for Hadoop workflow by describing the requirements not met by existing workflow systems in the large-scale Hadoop computing environments. In discussing the architecture of Oozie we explored some of the key capabilities of Oozie, how they meet the requirements, and identified several key areas for ongoing research. Then we analyzed the characteristics of the Oozie service in production, followed by some experimentation to quantify the scaling limiters for Oozie. Finally, we discussed some of the areas in which Oozie can mature. Oozie provides Yahoo and other organizations with major advantages in security, scalability and operability for Hadoop-based applications.

8. REFERENCES:

[1] Apache Hadoop. (n.d.). Retrieved February 24, 2012 from http://hadoop.apache.org/

[2] Apache Oozie. (n.d.). Retrieved February 24, 2012 from http://incubator.apache.org/oozie/

[3] Oracle Big Data Appliance. (n.d.). Retrieved February 10, 2012 from http://www.oracle.com/us/products/database/big-data-appliance/overview/index.html

[4] Cloudera's Distribution Including Hadoop. (n.d.). Retrieved February 10, 2012 from http://www.cloudera.com/hadoop-details/

[5] IBM InfoSphere BigInsights Information Center. (n.d.). Retieved February 10, 2012 from http://publib.boulder.ibm.com/infocenter/bigins/v1r3/index.jsp

[6] Deelman, E., Gannon, D., Shields, M., & Taylor, I. May 2009. Workflows and e-Science: An overview of workflow system features and capabilities, Future Generation Computer Systems, Volume 25, Issue 5, May 2009, Pages 528–540.

[7] Kepler. (n.d.). Retrieved February 10, 2012 from https://kepler-project.org/

[8] Taverna (n.d.). Retrieved February 10, 2012 from http://edutechwiki.unige.ch/en/Taverna_workbench

[9] Jianwu Wang, Daniel Crawl, Ilkay Altintas. Kepler + Hadoop : A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems. In Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science (WORKS09) at Supercomputing 2009 (SC2009) Conference, Portland, Oregon, USA, November 14-20, 2009

[10] Gurmeet Singh, Mei-Hui Su, Karan Vahi, Ewa Deelman, Bruce Berriman, John Good, Daniel S. Katz, and Gaurang Mehta. Workflow Task Clustering for Best Effort Systems with Pegasus. Mardi Gras Conference, Baton Rouge, LA, January 2008

[11] Dan Gunter, Ewa Deelman, Taghrid Samak, Christopher Brooks, Monte Goode, Gideon Juve, Gaurang Mehta, Priscilla Moraes, Fabio Silva, Martin Swany, Karan Vahi. Online Workflow Management and Performance Analysis with Stampede, 7th International Conference on Network and Service Management (CNSM-2011), Paris, France, October 2011

[12] Weiwei Chen, Ewa Deelman, Workflow Overhead Analysis and Optimizations, 6th Workshop on Workflows in Support of Large-Scale Science (WORKS 11), Seattle, Washington, November 14th, 2011.

[13] The Azkaban Project. (n.d.). Retrieved February 24, 2012 from http://github.com/azkaban/azkaban

[14] Apache Pig. (n.d.). Retrieved February 24, 2012 from http://pig.apache.org

[15] Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., Reed, B., Srinivasan, S. & Srivastava, U. 2009. Building a high-level dataflow system on top of map-reduce: The Pig experience. In Proc. VLDB.

[16] Thusoo, A., Sarma, J., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P. & Murthy, R. 2009. Hive-A Warehousing Solution Over a Map-Reduce Framework, IN VLDB '09: PROCEEDINGS OF THE VLDB ENDOWMENT

[17] The Hive Project. (n.d.). Retrieved February 24, 2012 from http://hadoop.apache.org/hive/

[18] Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. 2010. ZooKeeper: Wait-free coordination for internet-scale systems. In Proc. USENIX Annual Technical Conference.

[19] Google App Engine Multitenancy. (n.d.). Retrieved February 10, 2012 from http://code.google.com/appengine/docs/java/multitenancy/overview.html

[20] Chong, F., Carraro, G., & Wolter, R. 2006. Multi-Tenant Data Architecture. Retrieved February 10, 2012 from http://msdn.microsoft.com/en-us/library/aa479086.aspx

[21] Wang, C. (2009, November 18). Cloud Security Front and Center [Web log comment]. Forrester Research. Retrieved from http://blogs.forrester.com/srm/2009/11/cloud-security-front-and-center.html

[22] Brodkin, J. 2008, July 02. Gartner: Seven cloud-computing security risks. *InfoWorld*. Retrieved from http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853

[23] Security Guidance for Critical Areas of Focus in Cloud Computing. (n.d.). Cloud Security Alliance. Retrieved February 24, 2012 from https://cloudsecurityalliance.org/research/projects/security-guidance-for-critical-areas-of-focus-in-cloud-computing/

[24] Zhang, K. 2009. Adding user and service-to-service authentication to Hadoop. Retrieved February 24, 2012 from https://issues.apache.org/jira/browse/HADOOP

[25] O'Malley, O. 2010. *Integrating Kerberos into Apache Hadoop* [PDF of PowerPoint slides]. Retrieved from the 2010 Kerberos Conference: Agenda and Slides site http://www.kerberos.org/events/2010conf/2010slides/2010kerberos_owen_omalley.pdf

[26] Secure Impersonation using UserGroupInformation.doAs 2011. (n.d.). Retrieved February 24, 2012 from http://hadoop.apache.org/common/docs/stable/Secure_Impersonation.html