International Journal of Internet, Broadcasting and Communication Vol.8 No.1 33-41 (2016)
http://dx.doi.org/10.7236/IJIBC2016.8.1.33

IJIBC 16-1-5

An Efficient Log Data Processing Architecture for Internet Cloud Environments

Julie Kim[1], Hyokyung Bahn[1*]

[1]*Dept. of Computer Science and Engineering, Ewha University, Seoul, 120-750, Korea*
kjulee114@gmail.com, bahn@ewha.ac.kr[*]

Abstract

Big data management is becoming an increasingly important issue in both industry and academia of information science community today. One of the important categories of big data generated from software systems is log data. Log data is generally used for better services in various service providers and can also be used to improve system reliability. In this paper, we propose a novel big data management architecture specialized for log data. The proposed architecture provides a scalable log management system that consists of client and server side modules for efficient handling of log data. To support large and simultaneous log data from multiple clients, we adopt the Hadoop infrastructure in the server-side file system for storing and managing log data efficiently. We implement the proposed architecture to support various client environments and validate the efficiency through measurement studies. The results show that the proposed architecture performs better than the existing logging architecture by 42.8% on average. All components of the proposed architecture are implemented based on open source software and the developed prototypes are now publicly available.

Keywords: *big data; log data; message queue; Internet cloud*

1. Introduction

Due to the recent shift of computing paradigm to Internet cloud, “Big Data” is becoming increasingly catching interest in both industry and academia realms. As the stored data volume becomes exponentially large with digital technologies, efficient management of big data is essential to improve the quality of service and also to provide appropriate information to both end users and system administrators. Among various categories of big data, log data is one of the important big data generated essentially from software systems that should be well managed to provide services efficiently. Furthermore, it also provides significant information to software developers in debugging or finding failure points. Computation resources or infrastructures dealing with log data are also becoming increasingly important issues. In existing computing environments, log data are occasionally used at particular situations. For example, they are used when the system failure occurs or users explicitly investigate information for specific purposes. As its name implies, log data itself does not have the functionality of producing useful information directly due to its large volume that cannot be managed efficiently with legacy infrastructures.

As the computing paradigm shifts to mobile cloud computing, the network traffic of software itself as well as its data increases rapidly in client-server architectures [19-22]. Particularly, as clients and servers communicate in a synchronous way, the performance degradation becomes significantly large. Log messages account for a large portion of the network traffic as they are generated very frequently, becoming potential

Manuscript Received: Nov. 25, 2015/ Revised: Dec. 15, 2015/ Accepted: Dec. 29, 2015
Corresponding Author: bahn@ewha.ac.kr
Tel: +82 2 3277-2368, Fax: +82 2 3277-2306.
Dept. of Computer Science and Engineering, Ewha University, 11-1 Daehyun-dong, Seodaemun-gu, Seoul, 120-750, Korea.

sources of the performance degradation.

Meanwhile, a lot of systems adopt relational databases to manage data storage and SQL is used for query processing. This infrastructure performs well in traditional systems where number of clients and data to be managed are limited. However, as the amount of data and the number of requests from clients are growing sharply, storage becomes the bottleneck of the traditional data processing systems. The latency perceived by clients increases dramatically, which cannot be endured any longer. To relieve this problem, a scalable infrastructure such as distributed file systems is needed as an alternative of legacy structures.

In this paper, we propose LogStore, a scalable log management system that consists of client and server side modules for efficient handling of log messages and aggregating them. The architecture handles large log messages from clients asynchronously and collects them through multiple message queues; then, it finally stores log data to the distributed file system in the servers. To support large and simultaneous log data from multiple clients, we adopt the Hadoop infrastructure in the server-side file system for storing and managing log data [2, 3]. This architecture handles log messages from clients asynchronously and saves them into Hadoop Distributed File System.

We design the proposed architecture in CentOS 5.5 and develop sample client applications to support HDFS (Hadoop Distributed File System) with Eclipse RCP (3.0v, Indigo). The applications at client-side produce log messages concurrently and send them to the server. Finally, LogStore displays them with the saved log files.

The remainder of this paper is organized as follows. Section 2 describes how LogStore sends log messages to a message queue in the server. Section 3 discusses the Hadoop infrastructure used to support the proposed architecture. In Section 4, we describe the implementation details of LogStore. Section 5 validates the feasibility of the architecture through measurement studies. Finally, we conclude this paper in Section 6 with discussing some future work.

2. Basic Architecture of LogStore

The basic architecture of our LogStore is depicted in Figure 1. Each application in the client-side generates log data and sends them to the message queue in the server through various paths and ports. Message queues provide asynchronous processing of each log message. Particularly, they do not have any effect on client states and work independently to each other. Paths and ports between message queues and clients can be configured by logging systems on the clients and message queues on the server. Server applications store log data in HDFS after proper handling. Specifically, a server application dequeues log messages from one of the message queues, and transfers them to the distributed file system to append or write to appropriate files. We use the HDFS as the file system of the proposed architecture. Details of the processing will be discussed in Section 3.

In our architecture, gathering and handling log messages is not related to client-sides as it is performed by server applications. Specifically, log message processing in the client-side does not degrade the performance of the client. This is because servers receive messages and handle them independently to client status. Message queues enable the stateless handling of log messages for both servers and clients. In the server, a message queue daemon and a server application dealing with log data are independent processes. This

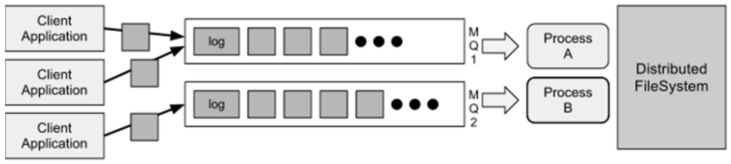


Figure 1 A basic architecture of LogStore.

separation provides some advantages in managing the overall system. For example, clients send log messages to a message queue although a server application may be in failure as sending log messages depends on the message queue daemon, not the state of the server application. Also, processing log messages is up to server applications, not message queues. The proposed architecture provides scalability in appending clients, message queues, and server applications. Moreover, it improves the availability of systems by enhancing the failover function as checking each layer is separated.

3. Store Log Data with Hadoop Infrastructures

Huge amount of log data should be managed efficiently while storing and processing not to influence the performance of other system components. In transaction-based database systems, large data delays the response time of query processing seriously. To relieve this problem, storing massive data in distributed storage and processing them in parallel is required. Hadoop is one of the efficient infrastructures that satisfies these requirements in distributed systems [2]. Hadoop is efficiently designed for sequential writes, contiguous appends, and a lot of reads, which fit well for the characteristics of log data. In the Hadoop distributed file system (HDFS), a large file is split into the same size chunks, and then stored to multiple HDFS nodes redundantly. Chunks are distributed to multiple nodes considering network topologies and load of each node. In this architecture, read requests for large files can be handled efficiently through concurrent reads from multiple HDFS nodes. Although some nodes fail during file operations, it can provide the failover capability by using alternative replicated chunks stored in other HDFS nodes. Our LogStore provides efficient data processing and fault tolerance by using Hadoop as a base infrastructure.

LogStore consists of four layers: a client application that generates log data, a message queue system that relays log data between clients and servers, a server application that handles delivered messages with file API, and HDFS that actually stores log data. Figure 2 shows each layer of LogStore. A log message generated from the client application is delivered to the message queue and processed by an appropriate server application. A server application handles the message and stores in the distributed file system based on Hadoop.

One of the important requirements in big data management is the fast extraction of meaningful information from stored data. This can be realized in the proposed LogStore as Hadoop supports the MapReduce programming model [9], which performs map and reduce operations repeatedly with (Key, Value) pairs. The results can be obtained promptly as each MapReduce operation is performed simultaneously on different processors. In addition, Hadoop eco-systems provide a lot of convenient tools like Pig [13] and Hive [4] to handle big data.

4. Implementations

In this section, we discuss the implementation details of LogStore. All components of LogStore are implemented with open-source softwares, and the source code and binary of our projects are also publicly available on Github [6].

Client modules are developed by Eclipse RCP (Rich Client Platform) and libraries with Java. An Eclipse RCP application in the client generates log messages and the client logging system sends them to the server message queue. The client logging system is organized with a logging framework in Java, Slf4j and Log4j.

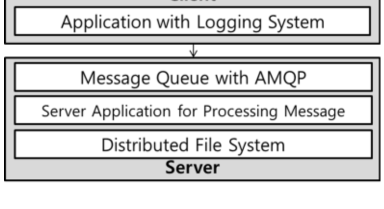

Client
Application with Logging System
Message Queue with AMQP
Server Application for Processing Message
Distributed File System
Server

Figure 2 A detailed architecture of LogStore using Hadoop.

Slf4j consolidates log messages from various logging frameworks like commons-logging. Log4j formats messages and prepares to send them to the message queue. To implement a message queue in the server, we use RabbitMQ. Usually, message passing is performed by JMS (Java Message Service) in Eclipse RCP because it is built in Java and supports java applications efficiently. However, as JMS supports only homogeneous architectures, both publishers and subscribers should be supported by Java. To provide more general frameworks for heterogeneous client-server architectures, we use AMQP (Advanced Message Queuing Protocol) instead of JMS. RabbitMQ is the representative messaging software that supports AMQP and it also exhibits good performance as it is implemented by the Erlang language. Server applications are developed by the Flume node instance that connects message queues and HDFS. By using server applications, log messages are dequeued from the message queue and then stored into HDFS by file rolling. Table 1 and Figure 3 show the outline of each layer. We consider two kinds of implementations. The first sends log messages through client file systems while the second sends them directly to the server message queue without passing through the client file systems.

Table 1. Mapping of Architecture Layers and Software.

Client Logging System	Log4j with SLF4j
Message Queue	RabbitMQ
Server Application	Flume
Distributed File System	HDFS

4.1 File Tailing Architecture

In the legacy architecture, applications store log data to their local file systems. Although it needs more file I/Os, we initially use the file tailing architecture that makes use of the legacy logging system. Our file tailing architecture retrieves log data from local file systems and sends to the server message queue. This implementation needs an additional Flume node instance running on the client machine. Specifically, log messages generated by client applications are appended to the log file in the local file system, and the Flume node instance in the client machine performs the tailing of the log file. Then the generated messages are sent to the message queue in the server. Communications between Flume and RabbitMQ are performed by AMQP [10]. At the same time, a Flume node instance in the server machine dequeues data from the message queue and sends them to HDFS. Figure 4 shows the file tailing architecture of LogStore. Note that this supports multiple client applications with a configured logging system. If some clients are added, our logging system can send their log data to another added message queues to support scalability. Flume sink nodes are also appended to communicate with new message queues.

This basic architecture is then extended to large scale file tailing architectures as shown in Figure 5. It supports scalability by adding new message queues or Flume sink nodes to save log data to HDFS.

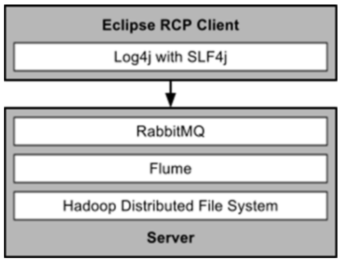


Figure 3 Concrete Composition of the LogStore Architecture.

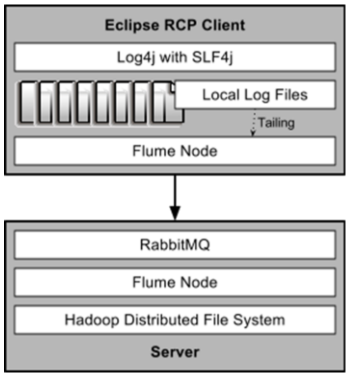


Figure 4 File Tailing Architecture of LogStore.

4.2 Log Appender Architecture

In the file tailing architecture, log messages are sent to the server via client local file systems. It needs additional write and read operations on the local file system and may degrade the performance of client systems. To relieve this problem, we develop a log appender of Log4j and AMQPAppender that does not store log data to local file systems but sends them directly to message queues. Paths or ports between servers and clients are configured by the same way of general logging frameworks. AMQPAppender is responsible for formatting and sending log messages to the message queue in the server. After sending messages, the message queue system in the server-side enqueues the log messages to a specific queue matched with the setup of AMQPAppender. Then, a Flume node instance in the server machine dequeues messages from the message queue that it needs to deal with and saves in HDFS. Messages are saved with file rolling periodically.

Figure 6 shows the architecture with the client-side log appender. Unlike the architecture shown in Figure 5, Flume nodes do not exist at client machines. Instead, a logging system in the client application sends log messages directly to message queues through Advance Message Queue Protocol. The Flume agent plays the role of the server application that dequeues log data from the message queue and saves them into HDFS.

Figure 7 depicts how LogStore can build scalable architectures with AMQPAppender and Flume agent applications. As shown in the figure, Flume agents running on the servers are mapped to server applications. Specifically, it consists of two Flume nodes that are sink and source processes [1].

4.3 AMQPAppender for log4j

In LogStore, client applications do not need to know how log messages are handled in the logging system.

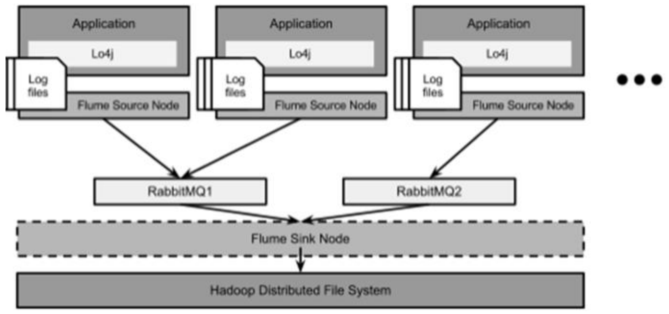


Figure 5 Large Scale of File Tailing Architecture

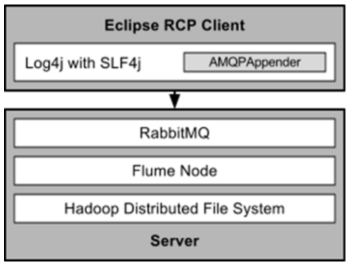


Figure 6 Log Appender Architecture of LogStore.

The application creates a logger from LoggerFactory of Slf4j and just publishes. Handling process is up to LogAppender of Log4j [11]. Though appenders supported by JMS (Java Message Service) already exist, they do not support heterogeneous jobs between clients and servers, and applications should be based on Java. Also, appenders of Log4j do not support AMQP yet. For this reason, we developed AMQPAppender supporting message queues with AMQP. Accordingly, when a log message is published, AMQPAppender sends it to the configured message queue via AMQP. The paths and ports can be changed with Log4j configurations.

5. Performance Evaluations

To assess the effectiveness of LogStore, we perform measurements with various client configurations. We implement client applications that generate log data to emulate various logging configurations. Our client applications show the current status of HDFS, and users can create, retrieve, and delete log files. Figure 8 shows the screenshot of Eclipse RCP client applications. It also shows metadata such as owner, permission, size, and created date of a file, and the retrieved contents of it by downloading. Console view below the file list displays the log messages generated during the execution of the application. All of the logs shown are sent to the message queue by the client logging system.

We measure the client performance with different logging architectures. Figure 9 compares the overhead of logging in terms of the application execution time. In the figure, No-logging represents the execution time when the system does not perform logging. Note that the execution time of other logging systems is displayed relative to that of No-logging. LogStore represents the proposed logging system that uses server-side message queues and Local logging represents the architecture that stores log data at local file systems. As shown in the figure, logging in the local file system incurs large overhead. Specifically, it takes 18 times longer latency than No-logging. This implies that logging is a serious performance bottleneck and optimization of logging performance is an important issue in modern software design.

There have been some attempts to relieve this problem. For example, SCM (Storage Class Memory) is adopted to perform logging more efficiently in DBMS [18]. However, it needs additional resources and

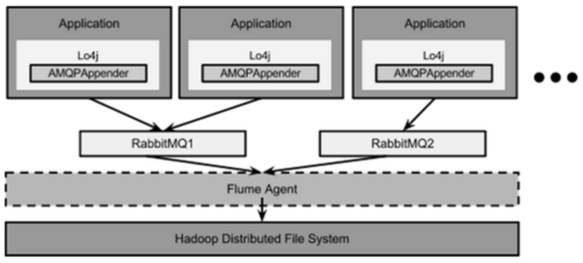
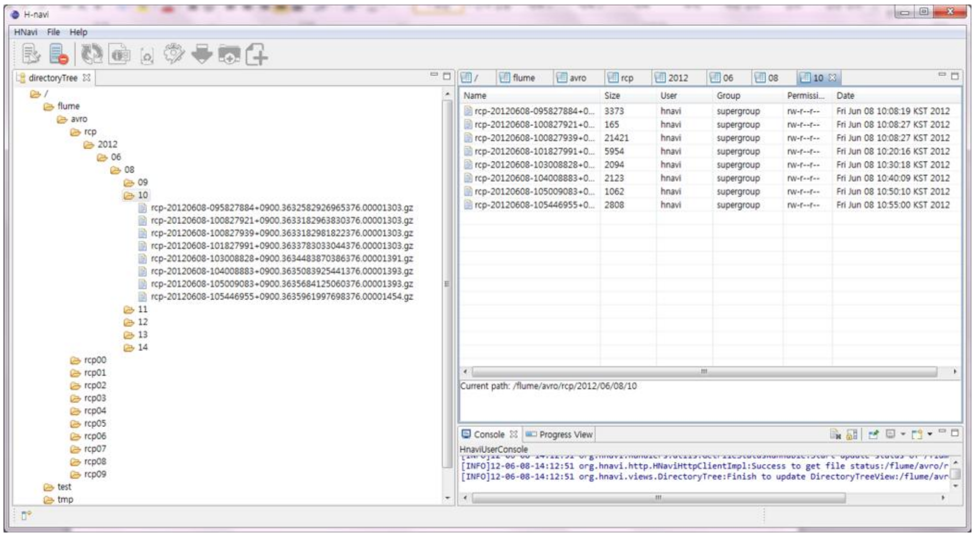


Figure 7 Large Scale of Log Appender Architecture



```
/
  flume
    avro
      rcp
        2012
          06
            08
              09
              10
                rcp-20120608-095827884+0900.3632582926965376.00001303.gz
                rcp-20120608-100827921+0900.3633182963830376.00001303.gz
                rcp-20120608-100827939+0900.3633182981822376.00001303.gz
                rcp-20120608-101827991+0900.3633783033044376.00001303.gz
                rcp-20120608-103008828+0900.3634483870386376.00001391.gz
                rcp-20120608-104008883+0900.3635083925441376.00001393.gz
                rcp-20120608-105009083+0900.3635684125060376.00001393.gz
                rcp-20120608-105446955+0900.3635961997698376.00001454.gz
              11
              12
              13
              14
      rcp00
      rcp01
      rcp02
      rcp03
      rcp04
      rcp05
      rcp06
      rcp07
      rcp08
      rcp09
  test
  tmp
```

Name	Size	User	Group	Permissi...	Date
rcp-20120608-095827884+0...	3373	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:08:19 KST 2012
rcp-20120608-100827921+0...	165	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:08:27 KST 2012
rcp-20120608-100827939+0...	21421	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:08:27 KST 2012
rcp-20120608-101827991+0...	5954	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:20:16 KST 2012
rcp-20120608-103008828+0...	2094	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:30:18 KST 2012
rcp-20120608-104008883+0...	2123	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:40:09 KST 2012
rcp-20120608-105009083+0...	1062	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:50:10 KST 2012
rcp-20120608-105446955+0...	2808	hnavi	supergroup	rw-r--r--	Fri Jun 08 10:55:00 KST 2012

Current path: /flume/avro/rcp/2012/06/08/10

```
[INFO]12-06-08-14:12:51 org.hnavi.http.HNaviHttpClientImpl:Success to get file status:/flume/avro/r
[INFO]12-06-08-14:12:51 org.hnavi.views.DirectoryTree:Finish to update DirectoryTreeView:/flume/avr
```

Figure 8 Eclipse RCP Client Application Screen shot of LogStore.

cannot be implemented directly to conventional computing environments. Thus, we aim to relieve it through software solutions. As shown in the figure, our LogStore does not degrade the client performance seriously compared to the legacy logging performed on local file systems by handling log data asynchronously through the message queue.

Figure 10 compares the throughput of different logging systems. Specifically, the number of logs handled per second is compared for the two logging architectures. As shown in the figure, using message queue performs better than local file logging significantly. Specifically, local file logging incurs serious performance degradation to client applications as the number of log messages increases. Also, it shows that LogStore with message queue relieves the overhead of logging and clients can send more log messages within a given time.

6. Conclusions

In this paper, we proposed a novel logging architecture with message queues and Hadoop infrastructure called LogStore. LogStore provides efficient and scalable log management systems for various client and server configurations. All software modules of LogStore are implemented with open source projects that can be adopted in each layer of the architecture. As Hadoop architecture is still in its evolution [7, 17], we can expect that our LogStore can be enhanced even more. We may also improve the performance of LogStore by analyzing the overhead of each layer, and then reconstruct some layers with new software components [8].

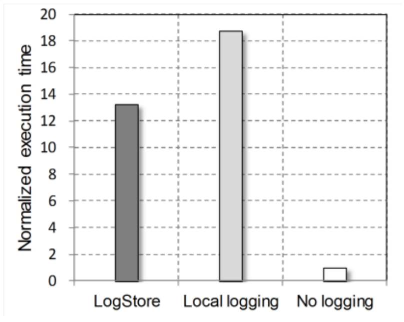


Figure 9 Overhead of logging

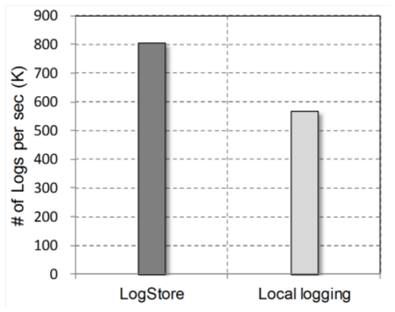


Figure 10 A Sample Performance result of LogStore

Another important aspect to consider is that the architecture can be enhanced if it supports event-driven processing. Event-driven processing is a new service area offering more appropriate information to users [12, 16]. The proposed architecture can be extended to support it in server applications. As a part of our future work, we will measure and compare the performance of our prototype implementations quantitatively [15]. It includes a new prototype architecture and various compositions of layers. Moreover, we plan to optimize software modules suitable for our architecture. We will also explore the advanced EDA (Event-Driven Architecture) fit for a large-scale distributed system. The proposed architecture has a potential for large distributed architectures applied to event-based processes, which have a lot of challenging issues.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2011-0028825). Hyokyung Bahn is the corresponding author of this paper.

References

[1] Apache Flume, http://flume.apache.org/

[2] S. Ghemawat, H. Gobioff, S. Leung, "The Google file system," Proceedings of ACM Symposium on Operating Systems Principles (SOSP), Bolton Landing, NY, pp.29–43, 2003.

[3] Hadoop infrastructure, http://hadoop.apache.org/

[4] Hive, http://www.hive.apache.org

[5] Gartner, The future of the Internet. http://www.gartner.com/technology/research/future-of-the-internet/

[6] H-navi, https://github.com/julnamoo/h-navi

[7] J. Horey, E. Begoli, R. Gunasekaran, S.H. Lim, J. Nutaro, "Big data platforms as a service: challenges and approach," Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), Boston, MA, 2012.

[8] J.P. Lozi, F. David, G. Thomas, J. Lawall, G. Muller, "Remote core locking: migrating critical-section execution to improve the performance of multithreaded applications," Proceedings of the USENIX Annual Technical Conference (ATC), Boston, MA, 2012.

[9] J. Zhao, J. Pjesivac-Grbovic, "MapReduce: the programming model and practice," Tutorials of the ACM SIGMETRICS Conference, Seattle, Washington, 2009.

[10] S. Vinoski, “Advanced message queuing protocol,” IEEE Internet Computing, vol. 10, no. 6, pp. 87–89, 2006.

[11] Log4j, http://logging.apache.org

[12] M. Migliavacca, I. Papagiannis, D.M. Eyers, B. Shand, J. Bacon, P. Pietzuch, “DEFCON: high-performance event processing with information security,” Proceedings of the USENIX Annual Technical Conference (ATC), Boston, MA, 2010.

[13] Pig, http://www.pig.apache.org

[14] RabbitMQ, http://www.rabbitmq.com/

[15] S. Appel, K. Sachs, A. Buchmann, “Towards benchmarking of AMQP,” Proceedings of the ACM International Conference on Distributed Event-Based Systems (DEBS), pp.99–100, 2010.

[16] T. Steiner, R. Verborgh, R. Walle, M. Hausenblas, J. Gabarró Vallés, “Crowdsourcing event detection in YouTube videos,” Proceedings of the Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE), 2011.

[17] X. Ye, M. Huang, D. Zhu, P. Xu, “A novel blocks placement strategy for Hadoop,” Proceedings of the International Conference on Information Systems (ICIS), pp. 3–7, 2012.

[18] R. Fang, H. Hsiao, B. He, C. Mohan, Y. Wang, “High performance database logging using storage class memory,” Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2011.

[19] B.P. Rimal, E. Choi, “A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing,” International Journal of Communication Systems, vol. 25, no. 6, pp. 796–819, 2012.

[20] Y. Lai, C. Lai, C. Hu, H. Chao, Y. Huang, “A personalized mobile IPTV system with seamless video reconstruction algorithm in cloud networks,” International Journal of Communication Systems, vol. 24, no. 10, pp. 1375–1387, 2011.

[21] I. Hsu, “Multilayer context cloud framework for mobile Web 2.0: a proposed infrastructure,” International Journal of Communication Systems, 2011; DOI: 10.1002/dac.1365.

[22] Y. Liu, Z. Chen, X. Lv, F. Han, “Multiple layer design for mass data transmission against channel congestion in IoT,” International Journal of Communication Systems, 2012: DOI: 10.1002/dac.2399.