International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 5, September- October 2017 ISSN 2278-6856

Improving the Performance of Resource Allocation inMultitenancy MapReduce Using YARN

Sunanda CP[1], Santhosh Kumar B[2]

[1]PG Student, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

[2] Assistant Professor, Computer Science and Engineering Dept, GPREC, Kurnool (District), Andhra Pradesh-518004, INDIA.

Abstract
Multitenancy MapReduce has earned the importance in the massive bigdata technology. In a multitenant MapReduce environment, multiple tenants with different demands can share a variety of Hadoop computing resources (e.g., network, processor, memory, storage and data) within a single Hadoop system, while each tenant remains logically isolated. This useful MapReduce Multitenancy concept offers highly efficient, and cost-effective systems without wasting Hadoop computing resources to enterprises requiring similar environments for data processing and management. In this paper, we are proposing an improved resource allocation approach supporting multitenancy features for Apache Hadoop, a large scale distributed system commonly used for processing big data using YARN. We initially implement the Hadoop framework focusing on "yet another resource negotiator (YARN)", which is mainly used for managing Hadoop resources, map reduce application runtime, and Hadoop user access controls in the latest version of Hadoop. We then identify the problems of YARN for supporting multitenancy and then derive the solution framework to solve these problems. Based on these requirements, we design the details of multitenant Hadoop. We also present the industrial multitenant Hadoop implementation that results to validate the bigdata access control and to evaluate the performance enhancement of multitenant Hadoop. The proposed multitenant Hadoop framework work is optimized for geographically distributed data centers considering the locations of data and users.

Keywords: Access control, big data, cloud, Hadoop, multitenancy, resource management, yet another resource negotiator (YARN), geographically distributed data centers.

1. INTRODUCTION
Multitenancy is an operating system level implemented concept that allows sharing resources in one single system among multiple independent tenants or users by providing every user to have his/her own logically independent computing environment [1]. For example, two tenants/users can use the same physical system for their own purpose with some minimal own configuration processes as if they were the only tenant using the computer system. There are lot of enterprises like Microsoft's Azure, Amazon AWS which are providing this type of multitenant systems. They distribute the same computer for various customers. A representative company providing such multitenancy at the application level is SalesForce.com. Using the multitenant concept it has developed a customer relationship management (CRM) application that can be customized online and provided a variety of customers with different requirement[2].

Multi-tenant applications serve different customers with one application instance. This architectural style leverages sharing and economies of scale to provide cost efficient hosting. As multi-tenancy is a new concept, a common definition of the word and related concepts is not yet established and the architectural concerns are not fully understood. This paper provides an overview of important architectural concerns and there mutual influences. Besides that, it defines multi-tenancy and differentiates it from several related concepts. Recently, there have been a number of efforts to support multitenancy on hardware infrastructure in addition to software solutions [3]–[11]. Note that we define the concept of a tenant as a user group with similar purpose in various forms such as a single user, a small group of users, or an enterprise. The definition is analogous with the user group in a Linux system; however, in our case, it must represent a more complex organization compared to Linux, and hence, we use the term tenant throughout the paper.

In this paper, we consider Apache Hadoop as a base platform for providing multitenant features. Many worldwide companies and research institutes use Hadoop for storing and processing large volumes of data [12]. Studies on big data and the Internet of Things have been conducted on Hadoop-based systems in many application fields such as medical treatment, transportation, weather, and astronomy. Because Hadoop is open-source software, its functions and installation processes are frequently different from version to version, causing compatibility issues with other softwares. Moreover, the complicated configuration task to setup a Hadoop cluster and the vulnerable security in managing the nodes are other problems of the current version of Hadoop (Hadoop 2.0) [13]. A Hadoop system is composed of a number of nodes rather than a single node, and thus, the maintenance cost would be considerably higher if a customized cluster for each tenant was required. Therefore, the development of multitenant Hadoop is essential to address the requirements of multiple tenants with a low cost and high efficient system by solving the current Hadoop problems mentioned above.

In this paper, we propose multitenant Hadoop by adding multitenant functions to "yet another resource negotiator (YARN)" and "Hadoop distributed file system (HDFS)".

YARN is a new component of Hadoop 2.0 that provides the resource management function optimized for a distributed system. Because re source management is a major constituent in providing the concept of multitenancy, we first address YARN. To begin, we identify three problems in the original Hadoop which is used in many big data platforms provided by Hortonwork [14], Cloudera [15], and other organizations.

Problem 1: Hadoop cannot provide multitenancy because itlacks an integrated management function of the etadata for tenants,resources,accesscontrollists(ACL),andotherobjects. We call this problem *the metadata integrationand management problem*.

Problem 2: The real time reflection of information change is not possible in the current version of YARN because the current metadata are managed in a number of extensible markup language (XML) files. We call this problem *the real-time resource managementproblem.*

Problem 3: The current version of YARN cannot efficiently manage the cluster resources in a distributed system because it can only control limited resources (CPU and memory), and itis difficulttosupportfine-grainedresourceallocationbytenantor job. We define this problem *the multiple resource management problem.*

Problem 4: In current enterprise environment, no proper solution is available for multitenant Hadoop which is distributed across the globe.

The proposed multitenant Hadoop is designed and implemented to solve the above four problems, thereby significantly improve the performance and applicability of multitenancy Hadoop. This is the major contribution of this paper. The purpose of this paper is to present the appropriate solutions with their design and implementation for multitenant Hadoop as presented in Table 1. By applying the proposed solutions to YARN and HDFS of Hadoop 2.0, we can solve the problems relating to security vulnerability and tenant management. We can also provide the fine-grained resource allocation and enhanced resource management with real-time control, ultimately rendering multitenancy in Hadoop. With the proposed system, each tenant can have his/her own logically isolated computing and storage environment on one shared physical resource cluster. It obviously reduces the maintenance cost significantly and improves the efficiency of Hadoop. The contribution of the paper is as follows. First, we propose the concept and the requirements of multitenancy in Hadoop. Then, we analyze Hadoop for supporting multitenant functions and formally define three problems for providing multitenancy in Hadoop. By presenting a solution for each problem, we propose the novel design of multitenant Hadoop. Finally, we implement the proposed multitenant Hadoop and demonstrate its validity and superiority with real experimentation. To the best of our knowledge, we believe that our research is the first concrete work addressing the multitenant property in Hadoop. The rest of this paper is organized as follows. Section II describes related work and explains major research trends. Section III derives improvements and requirements for multitenancy by analyzing Hadoop. Section IV presents the proposed Multitenant YARN to be proposed and implemented. Section V presents the experimental evaluation of our implementation. Section VI summarizes and concludes the paper with future work.

A. Multitenancy

As mentioned in the introduction, multitenancy is technology that allows different tenants to share computing resources in a single system as if each is the only user of the system. With the growth of cloud computing technology, the coverage of multitenancy has been gradually broadened to support sharing at all layers of computing systems such as platforms, DBMSs, system infrastructures and applications. Many studies have been performed on multitenancy; we use the reference architecture defined by Gartner as illustrated in Fig. 1 [16]. Fig. 1 presents several multitenancy models according to the degree and type of resources that can be shared across ten ants. The first shared-nothing model, which means single tenancy, provides each tenant with a separate system.

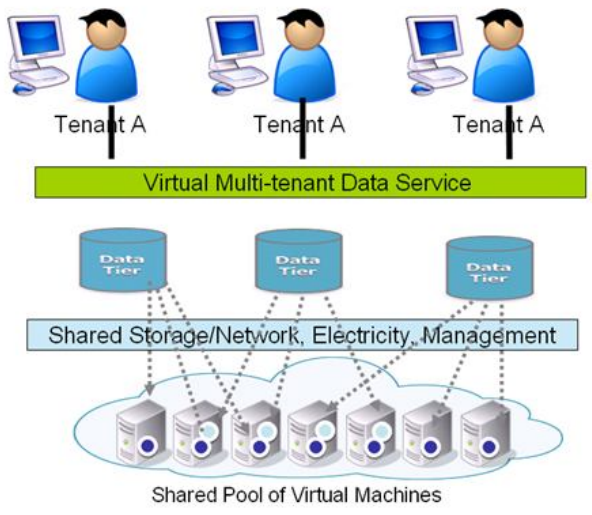
Tenant A
Tenant A
Tenant A
Virtual Multi-tenant Data Service
Data Tier
Data Tier
Data Tier
Shared Storage/Network, Electricity, Management
Shared Pool of Virtual Machines

Figure 1: Example for Multitenant System

The subsequent models provide “other parts” or everything-shared systems from infrastructures to applications, depending on the specific architecture. In general, the coverage of the system infrastructure is analogous to Infrastructure as a Service (IaaS) in cloud computing, the coverage of the database and application platform is Platform as a Service (PaaS), and the coverage of application logic is Software as a Service (SaaS). Virtualization and access control for security are considered as core technologies that require different approaches according to the characteristics of the models. Fig. 1 identifies the various models by sharing coverage; however, the majority of multitenancy research efforts focus primarily on a shared-everything system that can provide a logical independent computing environment to each user at low cost by reducing the inefficiencies caused by system installation and management for each user.

Table 1: Major feature trends on multitenancy

Approach	**Security Patterns**	**Extensibility Patterns**	**Scalability Patterns**
Separate Databases	Trusted Database Connections Secure Database Tables Tenant Data Encryption	Custom Columns	Single Tenant Scaleout
Shared Database, Separate Schemas	Trusted Database Connections Secure Database Tables Tenant Data Encryption	Custom Columns	Tenant-Based Horizontal Partitioning
Shared Database, Shared Schema	Trusted Database Connections Tenant View Filter Tenant Data Encryption	Pre allocated Fields Name-Value Pairs	Tenant-Based Horizontal Partitioning

B. Apache Hadoop and YARN

Apache Hadoop, a large open-source project, is a distributed processing framework for the management and analysis of large volumes of data, which has been a technology trend in the field of IT with big data issues worldwide. Hadoop is largely classified into version 1.x consisting of HDFS and MapReduce, and version 2.x, which incorporates YARN into version 1.x.

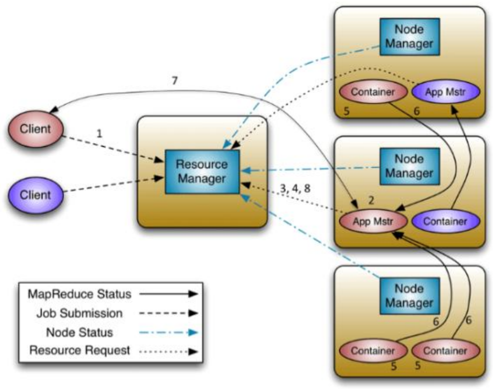

Figure 2: Overall architecture and execution flow of YARN

In the version 1.x, MapReduce is responsible for the data processing and resource management, whereas in version 2.x, the resource management is removed from MapReduce and addressed by the new component YARN. Because MapReduce in Hadoop 1.x performs both cluster and resource management and job processing management, severe bottlenecks occur in a large cluster of more than 4,000 nodes [17]. Further, because the number of map and reduce tasks that can be run on a single node is limited, even though some resources are available on that node, they may not be utilized [17]. To solve this problem, Yahoo! designed is YARN as an improvement of MapReduce. The purpose of YARN is to remove bottlenecks that occur in the existing system, by separating the resource management from the job execution management of MapReduce, and to provide a general-purpose computing environment for running a variety of applications in addition to Map Reduce. YARN consists primarily of Resource Manager, Node Manager, and Application Master; Fig. 2 illustrates the overall architecture and flow based on these components. Resource Manager, a kind of master server responsible for managing Hadoop internal resources, identifies the available resources in a cluster and allocates them to applications. Then, NodeManager, a module for managing the data processing work performed on each node, acts in a slave role to assist ResourceManager in the monitoring of the status of each node in a cluster. Finally, ApplicationMaster, a module executed by each application, monitors the application status and reports it to ResourceManager.

The client submits an application and necessary information to ResourceManager.

Resource Manager identifies an application node that can run the application submitted by the client and executes Application Master on that node. When Application Master runs normally, it registers itself to Resource Manager to send and receive information. Application Master makes a request to Resource Manager for a container to run the application. When Resource Manager assigns a container, Application Master directs the NodeManager with the corresponding container to execute the container and delivers the necessary information to execute the application. The application running in the container delivers the execution and status information to ApplicationMaster. While the application is running, the client directly communicates with ApplicationMaster to obtain information on the application execution. When the application completes, Application Master deletes the related information from Resource Manager and terminates itself. Then the container assigned to the application becomes available to other jobs. The basic execution flow of YARN shown in Fig. 2 is as follows [18]. In this paper, we modify and complement this flow to provide multitenancy functionality.

C. Hadoop Partitioning Skew

In the state-of-the-art MapReduce systems, each map task processes one chunk of the input data, and generates a sequence of intermediate key-value pairs. A hash function is then used to partition these key-value pairs and distribute them to reduce tasks. Since all map tasks use the same hash function, the key-value pairs with the same hash value are assigned to the same reduce task. During the reduce stage, each reduce task takes one partition (i.e., the intermediate key-value pairs corresponding to the same hash value) as input, and performs a user-specified reduce function on its partition to generate the final output. This process is illustrated in Fig. 1. Ideally, the hash function is expected to generate equal size partitions if the key frequencies, and sizes of the key-value pairs are uniformly distributed. However, in reality, the hash function often fails to achieve uniform partitioning, resulting into skewed partition sizes. For example in the InvertedIndex job [14], the hash function partitions the intermediate key-value pairs based on the occurrence of words in the files. Therefore, reduce tasks processing more popular words will be assigned a larger number of key-value pairs. As shown in Fig. 1, partitions are unevenly distributed by the hash function. P1 is larger than P2, which causes workload imbalance between R1 and R2. Zacheilas and Kalogeraki [3] presented the following reasons of partitioning skew:

Skewed key frequencies: Some keys occur more frequently in the intermediate data. As a result, partitions that contain these keys become extremely large, thereby overloading the reduce tasks that they are assigned to.

Skewed tuple sizes: In MapReduce jobs where sizes of the values in the key-value pairs vary significantly, even though key frequencies are uniform, uneven workload distribution among reduce tasks may arise.

In order to address the weaknesses and inadequacies experienced in the first version of Hadoop MapReduce (MRv1), the next generation of the Hadoop compute platform, YARN [15], has been developed. Compared to MRv1, YARN manages the scheduling process using two components: a) ResourceManager is responsible for allocating resources to the running MapReduce jobs subject to capacity constraints, fairness and so on; b) an ApplicationMaster, on the other hand, works for each running job, and has the responsibility of negotiating appropriate resources from ResourceManager and assigning the obtained resources to its tasks. This removes the single point bottleneck of Job- Tracker in MRv1 and improves the ability to scale Hadoop clusters. In addition, YARN deprecates the slot-based resource management approach in MRv1, and adopts a more flexible resource unit called container. The container provides resource-specific, fine-grain accounting (e.g., < 2GBRAM;1CPU>). A task running within a container is enforced to abide by the prescribed limits. Nevertheless, in both Hadoop MRv1 and YARN, the schedulers assume each reduce task has uniform workload and resource consumption, and therefore allocate identical resources to each reduce task. Specifically, MRv1 adopts a slot-based allocation scheme, where each machine is divided into identical "slots" that can be used to execute tasks. However, MRv1 does not provide resource isolation among colocated tasks, which may cause performance degradation at run-time [16].

2. DATASET PREPARATION AND DESCRIPTIONREQUIREMENTS ANALYSIS OF MULTITENANT YARN

A. Problem Analysis of the Original Hadoop

As identified in the Introduction, the original Hadoop and YARN do not include an integrated metadata management scheme (Problem 1), do not provide real-time resource management (Problem 2), and have a limited resource management scheme (Problem 3). In this section, we describe each of these problems in more detail. Problem 1, which is not only related to YARN but also related to overall Hadoop, makes it impossible to support multitenancy owing to the lack of an integrated metadata management scheme. The core metadata for multitenancy include information on users/tenants, components, ACLs, system resource allocation, and other resource information. Such metadata are important factors necessary to operate a Hadoop cluster and are stored in a number of XML files in the latest version of Hadoop. However, because the size of the metadata increases as with the number of users increase, it requires a considerable amount of time to read and write the XML files. Therefore, in this paper, we provide a novel design for the integrated metadata to efficiently manage the resource information. This is related to the solution of Problem 2, and consequently, must be considered to effectively use database functions for adding metadata management features to the original YARN Problem 2, as it relates to YARN, makes real-time resource management impossible. Because it uses XML files, we must restart the system to apply the metadata when they change. However, it is not only inefficient to restart Hadoop with a number of nodes whenever metadata are changed, but also inappropriate for a real-time service. Moreover, if multiple users are using a single Hadoop cluster together, more serious problems may result from the system restart. In this paper, we propose a metadata management scheme based on a database system as a solution and design it to support real-time resource management. Problem 3 can be further divided into two sub-problems. First, the original YARN manages only the CPU and memory as the distributed resources, resulting in many constraints on allocating and managing other important resources. In particular, for environments with many active users, it is essential to manage the network and disk resources. Therefore, the resource management scheme must be extended to address all system resources including CPU, memory, network, and disk. Second, the original YARN can only allocate resources for a job by the unit of ratio (%). However, detailed allocation per user is frequently required for resource management in a distributed system. To solve these problems, we design a multitenant scheduler to support resource allocation by user/job and value-based assignment, which is described in detail in Section IV.

B. Requirements Analysis for Multitenancy in Hadoop

In this section, we analyze the requirements for multitenant YARN based on the problems described in Section III. A. In Table 3,we summarize the requirements with the detailed features for each requirement.

REQ-META ,a requirement for Problems1and 2 in Section III. A, indicates that metadata management of users / tenants, Hadoop components, resources allocation, and the ACL of components is required. This requirement becomes a design base of the data management responsible for the overall information of Hadoop. A detailed explanation of the meta data for REQ-METAisasfollows:

- **User/tenant information**: This describes the informationon Hadoop users and the organizations to which each user belongs. Hadoop performs user authentication and authorization using the Linux account information (account name obtained by the Linux command, 'whoami'), which can be a security vulnerability if multiple users are sharing Hadoop. For example, if a malicious user discovers another user's Linux account name, s/he can access a Hadoop cluster by mimicking other accounts and cause a significant risk to the entire cluster if the account has full access privileges to file management and job execution in Hadoop. Further, owing

Table 2: Definitions of multitenant YARN requirements

Requiremnet	Definition	Associated problem
REQ-META	Integrated metadata to manage the information of users/tenants, resources, and ACL together of a Hadoop cluster	Problem 1 Problem 2
REQ-RTIME	Real-time reflection of user updates, ACL, and resource changes	Problem 2
REQ-MRES	Extension of the resource manage-ment to network and storage as well as CPU and memory	Problem 3
REQ-RALLO	Resource allocation per user/job	Problem 3
REQ-GEO	data distribution across the geography	Problem 4

to the lack of the ability to manage users and tenants, it is difficulttoreflectadiverseandcomplexorganizationalstructure. For this problem, we designed Hadoop based on Kerberosauthentication[19]toenhancethesecurityandmanage user/tenantswithvariousorganizationstructures.

- **Component information**: Components indicate the main contenttobemanagedwithinHadoop;examplesarefiles(directories), services, andjobs.
- **ACL information**: The ACL includes the access rights of a user/tenant to Hadoop components. Because the current YARNmanagestheinformationwithmultipleXMLfiles,dynamic system configurations are not feasible, and the only permission that can be granted is limited to enable/disable modes.
- **Resource allocation information**: Resource allocation indicates the resource information allocated to a user/tenant and job in YARN. An example is the amount of CPU, memory, network, and disk usage by auser/tenant.

REQ-RTIME is the requirement derived from Problem 2. REQ-RTIME requires that the modification of users, resources, and ACL information is to be reflected in a real-time manner. Considering the requirements from both REQ-META of the integrated metadata management and REQ-RTIME of the real-time metadata reflection, we must implement a database system having the advantages of easy data management, fast response time, and system stability for an efficient solution. Consequently, we design MetaDatabase

3. MULTITENANT YARN FRAMEWORK

A. OverallArchitecture

Fig. 3 presents the overall architecture and operation flow of the multitenant YARN framework proposed in this study. As shown in the figure, numerous modifications are required because clients' ability, Meta Database, Resource Manager, and Node Manager are added or updated, compared to the original YARN in Fig. 2. In Fig. 3, the functions underlined are modified or added to provide multitenancy inYARN.This changes reflect the requirements previously explained in subsection III-B; they are summarized in Tables 4 and 5.We briefly describe the figure and tables asfollows.

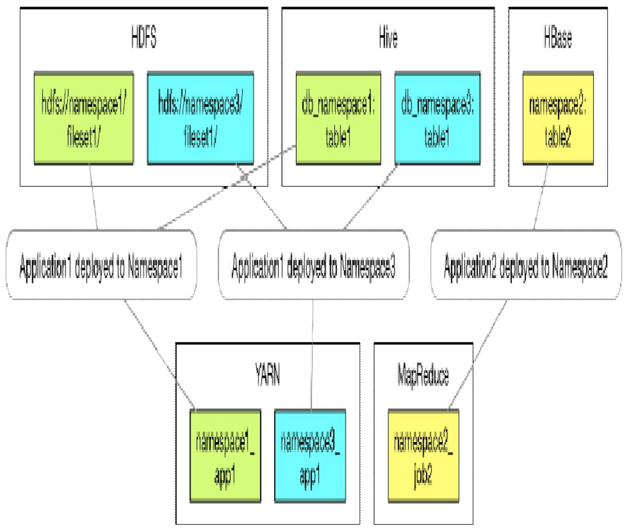
HDFS
hdfs://namespace1/ fileset1/
hdfs://namespace3/ fileset1/
Hive
db_namespace1: table1
db_namespace3: table1
HBase
namespace2: table2
Application1 deployed to Namespace1
Application1 deployed to Namespace3
Application2 deployed to Namespace2
YARN
namespace1_ app1
namespace3_ app1
MapReduce
namespace2_ job2

Figure 3: Multitenant Yarn

The primary difference from the existing system in Fig. 2 is the creation of MetaDatabase for the metadata management in Fig. 3. To provide the multitenancy function while sending and receiving information from/to MetaDatabase, modules in ResourceManager must be modified and added. Further more, Resource Manager must be modified to reflect metadata modification without a system restart, evenin the case where the administrator of the Hadoopclusteradds/updates/deletesvarious

In a multitenant environment, the greater the number of system users, the larger the information content in the system and metadata relating to the users and tenants. Similar to REQ-META, a previously defined requirement, because the metadata management scheme of the current version of Hadoop is difficult to support such multitenancy, the solution for the integrated metadata management is surely required for the multitenancy in a new Hadoop. Thus, in this paper we propose the integrated

MetaDatabase, which efficiently manages all metadata and reflects this metadata into the system quickly.

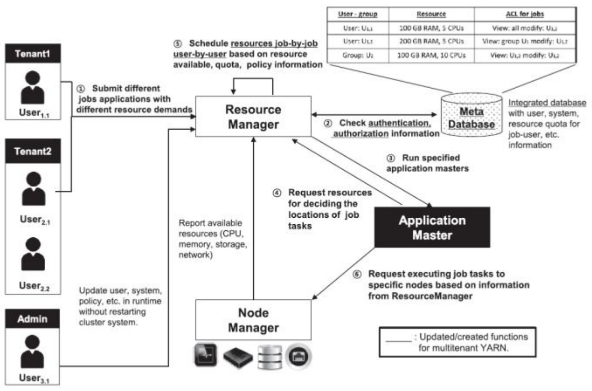
Tenant1
User1.1
Tenant2
User2.1
User2.2
Admin
User3.1
① Submit different jobs applications with different resource demands
⑤ Schedule resources job-by-job user-by-user based on resource available, quota, policy information
Resource Manager
② Check authentication, authorization information
Meta Database
Integrated database with user, system, resource quota for job-user, etc. information
User - group | Resource | ACL for jobs
User: U1,1 | 100 GB RAM, 5 CPUs | View: all modify: U1,2
User: U1,2 | 200 GB RAM, 5 CPUs | View: group U1 modify: U1,2
Group: U2 | 100 GB RAM, 10 CPUs | View: U1,2 modify: U1,2
③ Run specified application masters
④ Request resources for deciding the locations of job tasks
Application Master
Report available resources (CPU, memory, storage, network)
⑥ Request executing job tasks to specific nodes based on information from ResourceManager
Update user, system, policy, etc. in runtime without restarting cluster system.
Node Manager
: Updated/created functions for multitenant YARN.

Figure 4 Multitenant YARN Framework

(1) **User and tenant information**: This information is divided into a user table (USER), a tenant table (TENANT), and a user-tenant mapping table (USER_TENANT_MAP). Each user can be identified by ID_USER, and each tenant by ID_TENANT. Resource allocation and authorization in the system are performed through ID_USER and ID_TENANT. This many-to-many (N:M) mapping information between user (ID_USER) and tenant (ID_TENANT) is stored in the user-tenant mappingtable.

(2) **Hadoop service information**: Service - related information in Hadoop consists of a service table and a service ACL table. In Hadoop,there exist many kinds of services necessary for executing job sorusing the system.By managing the information on these services with ID_H_SERVICE in the service table, and granting authorization peruser/tenant in the service ACL table,un necessary access can be fundamentally prevented.

(3) **Job information**: This information created within Hadoop consists of a job table (JOB) and job ACL table (JOB_ACL). The job table is also used to allocate resources per user/job later. The job ACL table is necessary for access control on a job per user/tenant in the same manner.

(4) **Resource allocation information**: This information consists largely of a CPU quota table (CPU_QUOTA), a memory quota table (MEMORY_QUOTA), a disk quota ta ble(DISK_QUOTA), and a network quota table (NETWORK_QUOTA). Each quota table manages the allocated resource information by users in ID_USER, tenants in ID_TENANT, and jobs in ID_JOB, and has a different column structure according to the resource characteristics. When YARN actually executes jobs, it obtains the resource allocation values mapped to user/tenant/ job from these tables and applies them to the cluster.

B. Design of Multitenant Resource Manager

This section analyzes Resource Manager in Hadoop 2.x and proposes multitenant ResourceManager by including improvements. Resource Manager is responsible for managing job s submitted by YARN, scheduling resources, and monitoring nodes in a cluster. This paper enhances Resource Manager to support multitenancy,as indicated in Fig.5.To satisfy the three requirements, REQ-META, REQ-RTIME, and REQ-RALLO, multitenant quotas and access development, is necessary to provide multi-resource allocation by users and jobs. Components such as ClientService, Admin Service, and Web Service for communicating with the client must be modified to obtain the metadata from MetaDatabase. Another component groups for managing job must also be modified to enhance the performance and availability of the system and to synchronize the meta data. A detailed explanation of the senewly added or modified components is presented in Figure-5.

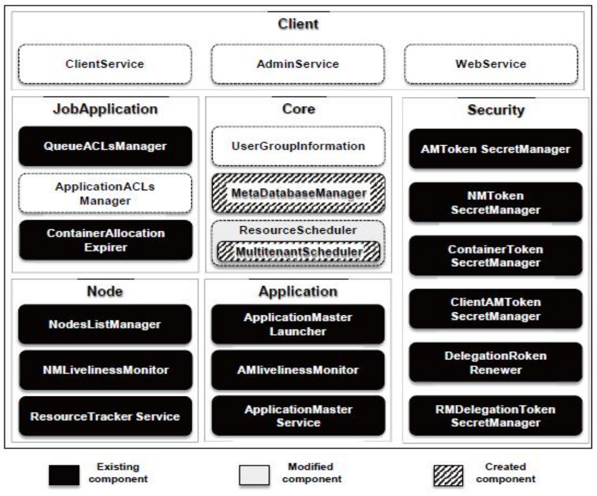

Figure 5: Configuration of experimental cluster.

C. Design of Multitenant Node Manager

This section presents the design of YARN Node Manager to support multitenancy. Node Manager of YARN, as explained in Fig. 2, executes a job application in each container, measures the resources available in the current node and informs the measurements to Resource Manager.Node Manager is closely related with REQ-RALLO,and to satisfy the requirement , the original YARN must be modified to allowextendibility of the types of resources allocated by Resource Manager.

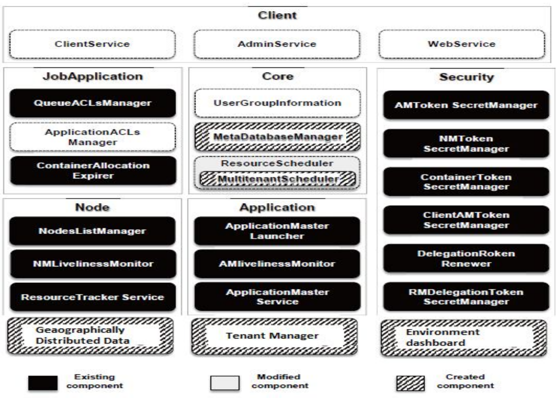

Figure 6 Multitenant Node Manager

Fig.6 shows multitenant Node Manager designed in this study. A sindicated in the figure,multitenant Node Manager has the same component structure as the existing one; however, it must be re-designed by modifying or extending each internal component for interoperation with theintegratedMetaDatabase.

Table 3: Hadoop and YARN components

Component	Functions
Containers Launcher	- Responsible for container execution within a node - Extended design of the existing component to add types of manageable resources (CPU, memory, disk, and network) and to reflect the resource information to the container at the start
Containers Monitor	- Monitoring container resources allocated by ResourceManager - Modified to allow monitoring CPU, memory, disk, and network
NodeStatus Updater	- Reporting node status to ResourceManager - Modified to deliver all information on CPU, memory, disk, and network
WebService	- Modified to obtain the information on running applications and extended ontainer resources - Provide the integrated Meta-Database to Web UI

4. EXPERIMENTAL EVALUATION

To demonstrate the validity and practical use of the proposed multitenant Hadoop,we performed an actual implementation by modifying Hadoop 2.6.0. In this section, we present the results of the experimental evaluation on the implemented multitenant Hadoop. The major goals of the experiments were as follows:

(1) To validate the Meta Database-based access control, (2) to examine the operation of resource allocation control, and (3) to demonstrate the performance superiority of multitenant Hadoop.

A. ExperimentalSetup

In the experiment, we configured an 10-node Hadoop cluster as illustrated in Fig. 7. Each node had four processors with a 2.4 GHz CPU, 3 GB RAM, 1 GB Ethernet NIC, and 1TB HDD. All nodes executed on unix mint, the proposed multitenant Hadoop, and Java 1.6.0. The first six nodes acted as Hadoop slaves (DataNode for HDFS and NodeManager for YARN). Node 7 was a Hadoop master (NameNode for HDFS and ResourceManager for YARN).

B. Experimental Results andAnalysis

Experiment1) We verified the performance of Meta Data based access control in the proposed prototype. To start the experiment, we first uploaded the file set to HDFS. Next, we randomly created a fixed number of ACL entries for the file set. Then, we restarted the cluster to record the bootstrapping time of the cluster. Note that this time was calculated from the time we started the HDFS cluster until NameNodeswitched into the normal working mode. Then, we performed random commands including reading /modifying ACL information and recorded the corresponding time. We repeated this process several times with an increasing number of ACL entries. As indicated in Fig. 8, the experimental results confirmed that the bootstrapping time of HDFS was completely independent from the number of ACL entries. They indicated that as additional metadata entries were added into the cluster, additional time was required by the cluster to perform the commands. However, this limitation can be solved by applying data base performance improvement methods based on a cluster or cache mechanism. Note that,because in this experiment the number of ACL s entries was changed dramatically, randomly accessing the ACL entries resulted in several major inflection points as shown in Fig. 8 for read and write request cases. This indicates the significant impact of the metadata information on the performance of Hadoop.

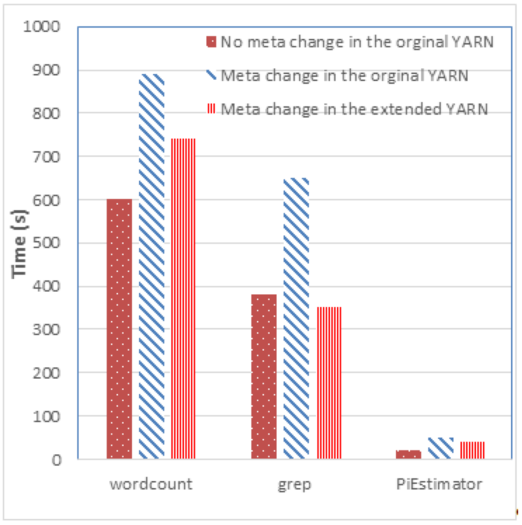

Figure 7: Performance of multitenant YARN prototype and original YARN

Experiment2) We evaluated the performance of the real-time reflectionofmetadatamodification.We compared the execution time of MapReduce applications (Word Count,GrepandPi Estimator) available in Hadoop source code in three cases:1) There was no metadata modification during job execution in the original YARN; 2) there was metadata modification during jobexecution in the original YARN; and 3) there was metadata modification during job execution in the proposed YARN. The three applications were representative of the processing encountered inatypicalmultitenantenvironment.WhiletheWordCountpro gram which calculates the number of occurrences of eachword in a text file and the Grep program which searches for matched lines in a text file based on regular expression matching are I/O bound and CPU-light because there are no expensive calculationsbeingexecuted.Thepi-estimatorprogramwhichcalculates an approximation using the Monte Carlo estimation method involves a pure computational workload with virtually no synchronization/communication overhead.

Experiment 3) Weexamined the performance of the multitenant YARN prototype with different resource requirements. Forthis,weconstructedan I/O intensive Map Reduce job,where each map task read files from HDFS and w rote files back tothe local disk of the assigned node. This allowed us to verify both disk and network control aspects of MultitenantScheduler. We compared the performance of the proposed MultitenantScheduler with the original CapacityScheduler of YARN by measuring the execution time of the MapReduce jobs for different users. We repeated the experiment ten times and calculated the average as the complete time. The experimental results are illustrated in Fig. 10. We note that, in all cases, the job execution time of the proposed MultitenantScheduler was less than that ofCapacityScheduler. The reason for the superiorperformance that the proposed Multitenant Scheduler with the awareness of available disk and network resources could assign tasks to the more appropriate nodes that satisfied to tally the resource requirements of the Map Reducejob

Experiment 4): We have evaluated the experiment of geographically distributed data in multitenant hadoop. For this, we have taken three clusters, one from India, second one from Canada and the third cluster from Singapore. There are nine tenants computed in three clusters, Table 3 presents the sub job sizes in each cluster, which is the number of tasks for jobs. Each cluster has one computation source, each cluster serves one task per second. Table 2 shows the approach of First Come First Scheduling (FCFS) approach across three clusters. Job order of tenants with FCFS is A ->B -> C, x axis represents the queue length (number of tasks to be served). If we use this FCFS scheduling, the average job completion time will be 13.3 s. However, we find that tasks of job A at Cluster2 has a higher completion time, then we may delay job A in favor of other jobs with faster completion time at clusters.

Table 4 -job execution in various clusters from various tenants

Job Arrival	Tenants	Cluster 1	Cluster 2	Cluster 3
A	Tenant3, 5,8	3	8	21
B	Tenant 6,7,8	6	18	16
C	Tenant1,2,4	19	25	13

Completion time:

Job A: 25 s

Job B: 19 s

Job C: 27 s

Average: 23.3 s

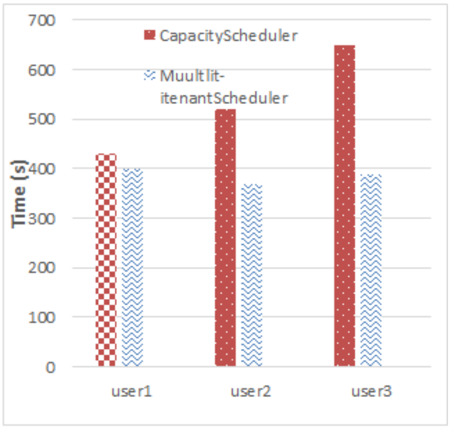

Figure 8: Performance of MultitenantScheduler and CapacityScheduler

Job A is submitted by tenants 3,5,8 and this job is distributed across the clusters managed by YARN. Job B is submitted by Tenants 6,7,8 and this job is also executed in all the available clusters but the execution time takes more than the earler job because the clusters already executing the previous jobs. So here FCFS concept is used to attend the jobs execution. In geographically distributed multitenant hadoop systems, tenant will be given to access for every cluster based on the availability and distance. So Tenants will get the more process time than the stand alone clusters.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an improved approach for multitenant Hadoop to access the shared environment across the globe with high performance. the Performance of Resource Allocation in Multitenancy MapReduce is increased by using YARN and distributed unix environment which is an extension of Hadoop focusing on YARN that could provide multitenant functions. The major contributions of this paper can be summarized as follows. First we thoroughly analyzed the reasons that made it difficult for Hadoop 2.x to provide multitenancy and clearly defined four problems. Then, we derived the five requirements necessary to support multitenancy and resolve these problems. Next, we presented an overall multitenant YARN framework to satisfy the requirements and proposed a detailed component structure and design scheme based on the actual Hadoop source code. Finally, through an actual implementation and experiments, we confirmed the validity and superiority of the proposed multitenant Hadoop. And then, we have implemented the multitenant mechanism with geographically distributed clusters to understand the computational speed across the clusters. So now the multitenant system can be used across the globe to access the hadoop and can submit the jobs. This is the first attempt to provide geographically distributed multitenancy features in Hadoop, and we feel that this is an excellent research result that significantly broadens the application range of Hadoop with low cost and more users (tenants).

As a future work, we will conduct additional work/research (experiments) to evaluate the proposed multitenant Hadoop. Because the experimental part of this paper focused mainly on multitenant resource management with distribution geographically, it did not reflect sufficiently the overall system performance. Therefore, after completing the multitenant modification, based on the new experimental results, we will attempt to complement our design to improve the system further.

References

[1]. Heesun Won, Minh Chau Nguyen, Myeong-Seon Gil, and Yang-Sae Moon, Advanced Resource Management with Access Control for Multitenant Hadoop, JCN, Dec 2015
[2]. ZhihongLiu,Qi Zhang, Reaz Ahmed,Dynamic Resource Allocation for MapReduce with Partitioning Skew, IEEE Nov 2016
[3]. M. Adrian and N. Heudecker, Hadoop 2015: The Road Ahead, Gartner Webinars, Nov. 2014.
[4]. J. Huang, D. M. Nicol, and R.H. Campbell, "Denial-of-service threat to Hadoop/YARN clusters with multi-tenancy," in Proc. IEEE Int'l Congress on Big Data, Anchorage, Alaska, June 2014, pp. 48–55.
[5]. S. Narayanan, Securing Hadoop, Packt Publishing Ltd., Nov. 2013.
[6]. V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in Proc. ACM SOCC, Santa Clara, California, Article No. 5, Oct. 2013.
[7]. (2017) Hortonworks: Open Enterprise Hadoop, [Online]. Available: http://hortonworks.com/
[8]. (2017) Cloudera, [Online]. Available: http://www.cloudera.com/
[9]. (2017) Apache Hadoop YARN-Concepts and Applications, [Online]. Available: http://hortonworks.com/blog/
[10]. Z. Tang, L. Jiang, J. Zhou, K. Li, and K. Li, "A self-adaptive scheduling algorithm for reduce start time," Future Gener. Comput. Syst., vol. 43, pp. 51–60, 2015.
[11]. Z. Liu, Q. Zhang, M. F. Zhani, R. Boutaba, Y. Liu, and Z. Gong,
[12]. "Dreams: Dynamic resource allocation for MapReduce with data skew," in Proc. IFIP/IEEE Int. Symp. Integrated Netw. Manage., May 2015, pp. 18–26.
[13]. (2016). Apache hadoop yarn [Online]. Available:http://hadoop.apache.org/docs/current/Hadoop/yarn/hadoop-yarn-site/YARN.html.
[14]. B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer network," IEEE Commun. Mag., vol. 32, no. 19, pp. 33–38, Sept. 1994.
[15]. V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in Proc. ACM SOCC, Santa Clara, California, Article No. 5, Oct. 2013.T. White, Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media, 2012

AUTHORE

Sunanda C.P. received the B.tech. degree in Computer Science from S K University and. She is currently studying M.Tech Computer Science in JNTU-Anantapur. Her research interests include multi-tenant system, cloud resource management, and big data analysis.

Santhosh Kumar received B.Tech (CSE) degree from JNTU and , M.S degree in Computer Science from Western Kentucky University.. He is currently an Assistant Professor in CSE Department in GPREC. Kurnool