Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2014, 6(6):2950-2954

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

Scientific data processing framework for Hadoop MapReduce

Kong Xiangsheng and Chen Jianbiao

Department of Computer & Information, Xinxiang University, Xinxiang, China

ABSTRACT

Scientific workflows produce large amounts of scientific data. Hadoop MapReduce has been widely adopted for data-intensive processing of large datasets. The Kepler system can support scientific workflows, high–performance and high-throughput applications, which can be data-intensive and compute-intensive. The paper presented a "Kepler + Hadoop" framework for executing MapReduce-based scientific workflows on Hadoop.

Keywords: MapReduce; Hadoop; scientific workflow; parallel processing

INTRODUCTION

The current scientific computing landscape is vastly populated by the growing set of data-intensive computations that require enormous amounts of computational as well as storage resources and novel distributed computing frameworks. On the one hand, scientific data centers, libraries, government agencies, and other groups have moved rapidly to online digital data access and services, drastically reducing usage of traditional offline access methods. On the other hand, practices for storage and preservation of these digital data resources are far from the maturity and reliability achieved for traditional non-digital media. On the industry front, Google Scholar and its competitors (e.g. Microsoft, CNKI, Baidu Library) have constructed large scale scientific data centers to provide stable web search services with high quality of response time and availability. Currently scientific workflows assist scientists and programmers with tracking their data through all transformations, analyses, and interpretations. A Scientific Workflow Systems is a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or a workflow, in a scientific application. In the future, scientific workflows will refer to the large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet. Typically, a feature of such collaborative scientific enterprises is that they will require access to very large data collections, very large scale computing resources and high performance visualization back to the individual user scientists. Current initiatives to effectively manage, share, and reuse ecological data are indicative of the increasing importance of data provenance.

Now scientific workflows are typically used to automate the processing, analysis, and management of scientific data. More and more automation tools, such as Kepler, Taverna, Vistrails, and many others have been designed in order to allow for scientific workflows to be created, executed, and shared among scientists and laboratories. They provide not only a way of tracing provenance and methodologies to help foster reproducible science and the publications of executable papers, but also a visual programming front end enabling users to easily construct their applications as a visual graph by connecting nodes together. By providing front-end visualizations and adaptations of shell scripts and manual steps, it is easier for scientists to do their work, especially when integrating grids and parallel processing or external databases.

STATE OF THE ART AND RELATED WORK

Scientific workflows produce huge amounts of scientific data from observations, experiments, simulations, models, and higher order assemblies, along with the associated documentation needed to describe and interpret the data, which are stored in large data warehouses in digital form [1]. Currently, more and more large-scale scientific problems are facing similar processing challenges on large scientific datasets which are a group of data structures used to store and describe multidimensional arrays of scientific data, where Hadoop could potentially help [2, 3]. Hadoop has become a widely used open source framework for large scale scientific data processing. In this paper I'm proposing that Kepler Scientific Workflow System and Hadoop MapReduce are better approaches and solutions for scientific data management.

MapReduce

MapReduce is a programming model for processing large datasets including scientific datasets. With the MapReduce programming model, programmers only need to specify two functions: Map and Reduce [4]. The map function takes an input pair and produces a set of intermediate key/value pairs. It is an initial transformation step, in which individual input records can be processed in parallel. The Reduce function adds up all the values and produces a count for a particular key. It is an aggregation or summarization step, in which all associated records must be processed together by a single entity. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. MapReduce functions are as follows.

Map:(in_key,in_value)$\rightarrow$$\{key_j, value_j \mid j=1\ldots k\}$
Reduce:(key, [$value_1$, $value_2$,…, $value_m$])$\rightarrow$(key, final_value)

The input parameters of Map are in_key and in_value. The output of Map is a set of <key,value>. The input parameters of Reduce is (key, [$value_1$, ..., $value_m$]). After receiving the parameters, Reduce is run to merge the data which were get from Map and output (key, final_value) [5].

Hadoop

Hadoop which is an open source implementation of the Google's MapReduce parallel processing framework is a more general distributed file system. The three Hadoop components that are analogous to Google's components described above are:
1. the MapReduce programming model
2. Hadoop's Distributed File System (HDFS).

HDFS is a flat-structure distributed file system that store large amount of data with high throughput access to data on clusters. HDFS has a master/slave architecture, and multiple replicas of data are stored on multiple compute nodes to provide reliable and rapid computations [6]. Its master node is called JobTracker or NameNode which is a simple master server, and TaskTrackers or DataNodes which are slave servers [7].

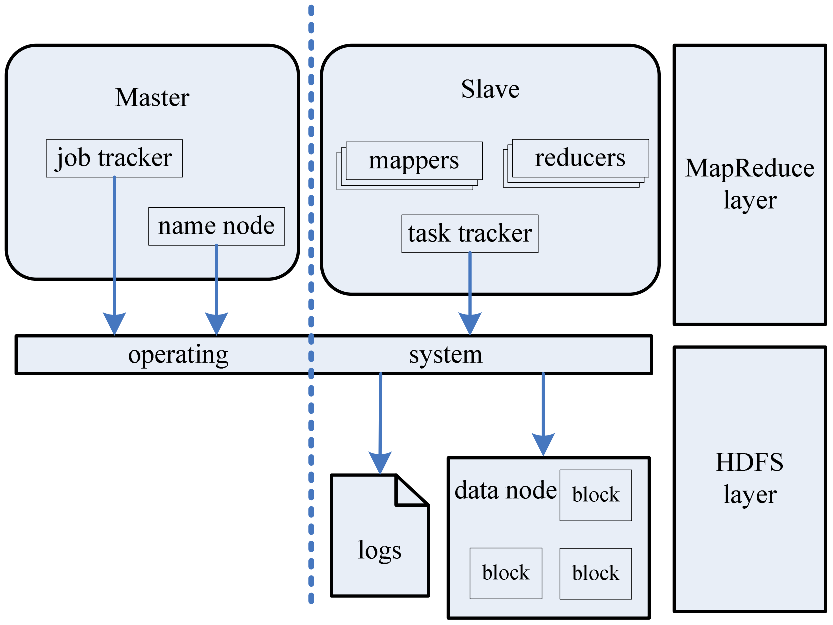

Figure 1. Architecture of Hadoop MapReduce

Kepler Scientific Workflow System

Kepler is a free software system for designing, executing, reusing, evolving, archiving, and sharing scientific workflows. Kepler is a type of "actor-oriented modeling" where actors are components that are designed to perform various processing tasks. Kepler actors perform operations including process and data monitoring, provenance information, and high-speed data movement solutions. Each actor has a set of input and output ports that provide the communication interface to other actors. Kepler's design actor can be seen as a "blank slate" which prompts the scientist for critical information about an actor, e.g., the actor's name, and port information. Kepler's web and Grid service actors allow scientists to utilize computational resources on the net in a distributed scientific workflow. Kepler includes database actors, e.g., DBConnect which emits a database connection token (after user login) to be used by any down-stream DBQuery actor that needs it [8].

Workflows can be organized visually into sub-workflows. Each sub-workflow encapsulates a set of executable steps that conceptually represents a separate unit of work. The Kepler system can support different types of workflows ranging from local analytical pipelines to distributed, high–performance and high-throughput applications, which can be data-intensive and compute-intensive [9]. Along with the scientific workflow design and execution features, Kepler has ongoing research on a number of built-in system functionalities, as illustrated in Fig.1 [10].

SCIENTIFIC DATA PROCESSING FRAMEWORK FOR MAPREDUCE

While Hadoop and the MapReduce paradigm can provide immense processing benefits for scientific users, there is also a considerable learning curve involved with using the Hadoop framework. The Kepler allows users to create workflows using a graphical user interface. Using Kepler, scientists can capture workflows in a format that can easily be exchanged, archived, versioned, and executed.

In the case of Kepler, MapReduce is implemented as an actor that can be added to workflows. In a workflow, actors have "ports" which either produce or consume data. Actors generally take data items in, process them, and then pass the results on to the next actor in the workflow. Data may take different paths through the workflow and can execute both serially and in parallel as shown in Fig.2. Kepler provides a good solution for users wanting to benefit from MapReduce without having to use it for every step in their processing.

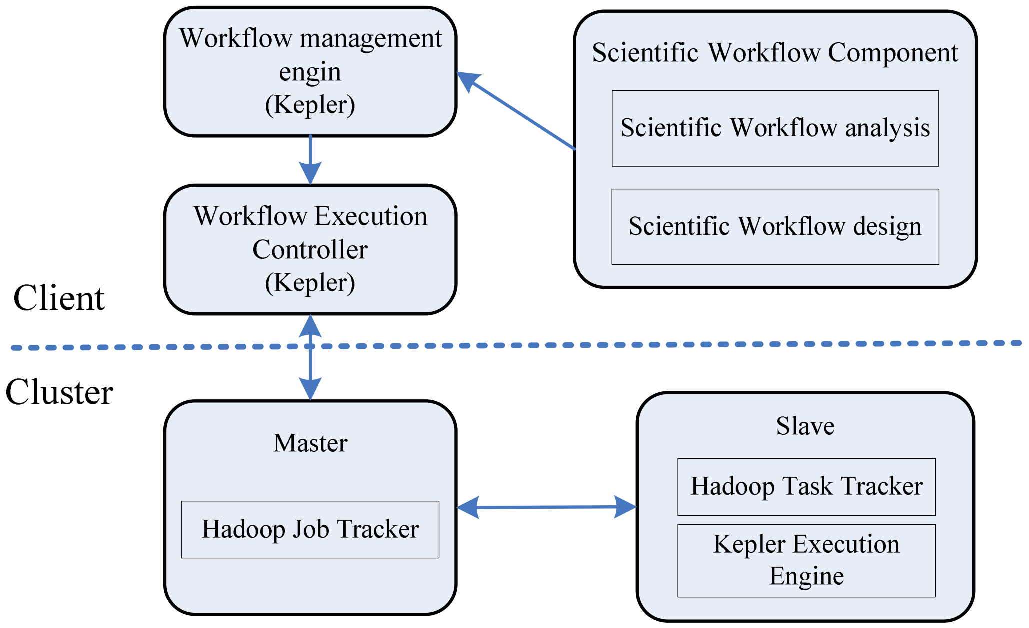
Workflow management engin (Kepler)
Scientific Workflow Component
Scientific Workflow analysis
Scientific Workflow design
Workflow Execution Controller (Kepler)
Client
Cluster
Master
Hadoop Job Tracker
Slave
Hadoop Task Tracker
Kepler Execution Engine

Figure 2. Scientific Data Processing Framework for MapReduce

Since Map and Reduce are two separate functions in the MapReduce programming model, Map and Reduce are treated as two independent sub-workflows in Kepler MapReduce actor [11]. Data written by some actors are read by actors on different nodes. Reduce tasks read outputs of Map tasks.

By default, data files are not stored in HDFS and instead are copied into HDFS from the filesystem before the MapReduce actor runs, so large changes to an existing workflow are not necessary. It is also possible to configure Kepler to use data that is already stored in HDFS, but then other actors would need to support HDFS as well if they need access to the data. The implications of copying files into HDFS before processing are not entirely clear, but could be a large bottleneck when working with massive datasets.

We implemented the Map and Reduce interface provided by Hadoop. When execution begins, the input data read by the Hadoop slaves will be transferred to the Map and Reduce subworkflows by our auxiliary input actors, such as the

MapInputKey and MapInputValue actor. Next, the Kepler engine will execute the Map/Reduce sub-workflows with the input data. Finally, our auxiliary output actors will transfer the output data of the subworkflows to the Hadoop slaves. The execution semantics for MapReduce actor execution in the Map and Reduce function are illustrated as shown in Fig.3.

```
map (key$_1$, value$_1$) {
    initialize Kepler execution engine for Map sub-workflow
    send key$_1$ to Kepler engine via MapInputKey actor
    send value$_1$ to Kepler engine via MapInputValue actor
    execute Map sub-workflow
    get (key$_1$, final_value) from Kepler engine via MapOutputList actor
    emit (key$_1$, final_value)
}
reduce (key$_2$, [value$_1$, value$_2$,⋯, value$_m$]) {
    initialize Kepler execution engine for Reduce sub-workflow
    send key$_2$ to Kepler engine via ReduceInputKey actor
    send [value$_1$, value$_2$,⋯, value$_m$] to Kepler engine via ReduceInputList  actor
    execute Reduce sub-workflow
    get value$_2$ from Kepler engine via ReduceOutputValue actor
    emit (key$_2$, final_value)
}
```

Figure 3. The execution semantics for MapReduce actor execution in the Map and Reduce function

The Hadoop MapReduce programming model will refer to a single master and multiple slave nodes. The master starts with sending such a message to each of the slaves. Then the master waits for any slave to return a result. As soon as the master receives a result, it will insert the result into the output array and provide further work to the slave if any is available. As soon as all work has been submitted to the slaves, the master will just wait for the slaves to return their last result. The master code would thus look like listed as shown in Fig.4.

```
master (){
    foreach slave {
        index , value = get_next_index_value_pair ();
        send (( index , value ), slave )
    }
    while ( work_available )
    {
        result , slave = receive_message_from_any_slave ();
        index , value = get_next_index_value_pair ();
        send (( index , value ), slave );
        output [ result . index ] = result . value ;
    }
    foreach slave {
        result , slave = receive_message_from_any_slave ();
        halt ( slave );
        output [ result . index ] = result . value ;
    }
}
```

Figure 4. Pseudo-code for the master

The slave code would thus look like listed as shown in Fig.5.

```
slave (){
    until ( halted ){
    work = receive_message_from_master ();
    send (( work . index , work . value * work . value ), master );
    }
}
```

Figure 5. Pseudo-code for the slave

CONCLUSION

It is clear to us that the traditional super computing centers consisting only of petascale computing resources are not sufficient to tackle the broad range of e-Science challenges. A reliable, data-intensive and compute-intensive, high–performance and high-throughput scientific workflow equipped with automation tools (i.e. Kepler) and parallel data analysis frameworks Hadoop MapReduce programming tool is needed.

REFERENCES

[1] Xiao Liu, The First CS3 PHD Symposium 2010, **2010**, 49-51.
[2] R. E. Bryant, Technical Report CMU-CS-07-128, Carnegie Mellon University, **2007**.
[3] J. Ekanayake; S. Pallickara; G. Fox, In Proceedings of the 4th IEEE International Conference on eScience (e Science 2008), **2008**, 277-284.
[4] Zhifeng Xiao; Yang Xiao, The First International Workshop on Security in Computers, Networking and Communications, **2011**, 1099-1104.
[5] DING Jian-li; YANG Bo, *International Journal of Digital Content Technology and its Applications*, **2011**(5), 236-243.
[6] B.Thirumala Rao; L.S.S.Reddy, *International Journal of Computer Applications*, **2011**(34), 28-32.
[7] Chen Zhang; Hans De Sterck; Ashraf Aboulnaga; Haig Djambazian; Rob Sladek, Lecture Notes in Computer Science, **2010**, 400-415.
[8] Ilkay Altintas; Chad Berkley; Efrat Jaeger; Matthew Jones; Bertram Ludäscher; Steve Mock, In 16th International Conference on Scientific and Statistical Database Management(SSDBM), **2004**, 345-367.
[9] Sangmi Lee Pallickara; Matthew Malensek; Shrideep Pallickara, On the Processing of Extreme Scale Datasets in the Geosciences, Handbook of Data Intensive Computing. **2012**, 521-537.
[10] Ilkay Altintas; Oscar Barney; Efrat Jaeger-Frank, Lecture Notes in Computer Science, **2006**, 118–132.
[11] Jianwu Wang; Prakashan Korambath; Ilkay Altintas1, 2011 IEEE World Congress on Services, **2011**, 212-215.