Building an Efficient Hadoop Workflow Engine Using BPEL

Jie Liu, Qiyuan Li, Feng Zhu, Jun Wei, and Dan Ye

Technology Center of Software Engineering
Institute of Software, Chinese Academy of Sciences
Beijing, China 100190
{ljie,liqiyuan09,zhufeng10,wj,yedan}@otcaix.iscas.ac.cn

Abstract. Big data processing and analysis techniques can guide enterprises to make correct decisions, and will play an important role in the enterprise business process. The Hadoop platform has become the basis of big data processing and analysis. To satisfy the needs of enterprises to develop data-intensive workflow based on Hadoop and integrate them into existing business processes, we build a Hadoop workflow engine named Pony based on BPEL model. The mapping method from Hadoop Workflow to BPEL process in three levels of the semantic model, deployment model, and execution model is presented. Pony uses a matured and stable BPEL engine to orchestrate Hadoop services. Pony implements a Hadoop job scheduler to collaborate with a BPEL engine to online schedule multiple workflows at runtime. This paper describes the design and implementation of Pony, and the experiment results demonstrate Pony can provide improved performance.

Keywords: MapReduce, Hadoop workflow, BPEL, Data intensive computing, Service oriented architecture.

1 Introduction

Big data processing and analysis techniques will play an important role in the enterprise business process. Hadoop [1] , the open-source implementation of MapReduce [2], is by far the most successful and popular data intensive cloud computing platform due to its remarkable features in simplicity, fault tolerance, and scalability. Users can submit MapReduce jobs consisting of a map function and a reduce function. A master node in the Hadoop cluster performs job scheduling, and distributes work to a number of slaves.

However, more frequently you will have many jobs with dependencies between them. For example, you might want to analyze the accumulated historical sales data to predicate the sale amount next month. You may first need to execute a MapReduce job to import the data to Hadoop, and then you need to execute a MapReduce job to normalize the data to prepare for the following data mining jobs. In this case, we need a Hadoop workflow engine to help us manage the job dependencies.

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 281–292, 2013.

In Fig.1, it shows a Hadoop workflow example. In this paper we assume Hadoop jobs include MapReduce job, Pig job, Hive job, HDFS job, System commands, Java application, and web service, which can be executed in Hadoop cluster. There are some workflow engines for Hadoop, such as Oozie [3], Cascading [4], Hamake [5], Azkaban [6] and CloudWF [7]. Unfortunately, this diversity of approaches generates a separated growth of best practices and methodologies. First, Hadoop workflow systems lack of a unified, standardized business process specification. Existing workflow systems use their own specification languages. Workflows generated by these systems are unable to communicate with each other, making the combination of business processes and enterprise applications very complex. It usually takes a long time for deployment and implementation, and integration with existing business process in the enterprise. Second, these workflow engines provide limited support for control logic. Existing Hadoop workflows only support these control routers: fork, decision, and join. This makes it hard to design complex data intensive workflow. Third, many data analysis processes need human interaction, we need to model these activities in the workflow model.

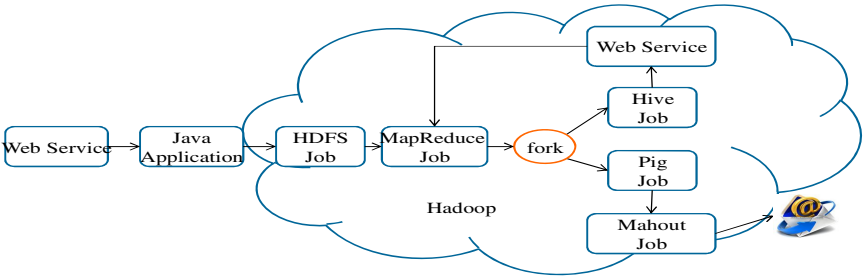

Fig. 1. Hadoop workflow example

BPEL provides a standard language for the specification of executable and abstract business processes [9]. BPEL supports rich control logic and support to model the human activities. BPEL is powerful enough to express complex data processing process. There are also some excellent open source BPEL engines. The benefits of the web service oriented framework are well appreciated. Therefore, our motivation is to adopt these matured workflow technologies to build an efficient and scalable Hadoop workflow engine. If we can define a Hadoop workflow with BPEL, we can build a Hadoop workflow engine quickly, and achieve high dependability by adopting a stable BPEL engine. Most enterprises have adopted BPEL as their workflow engine, so this can also make Hadoop workflow easily integrated into existing business process.

Hadoop workflow can be modeled as a Directed Acyclic Graph (DAG). BPEL is a type of block-structured flow definition languages. We should map a Hadoop workflow to a BPEL process at first. BPEL only can orchestrate web service. We propose to use wrapper service to convert a Hadoop job to a web service. For each job type, one wrapper service is deployed in the service container. The wrapper acts as an intermediary for the Hadoop job by exposing a Web service interface to the client.

The wrapper service is in chief of the job's deployment and execution and response the invoking from BPEL. In this way, we can avoid the service instance explosion.

Existing workflow systems such as Oozie provide no scheduler on workflow level. Instead, they just submit the ready jobs in the workflow to Hadoop cluster. However, the Hadoop job scheduler knows nothing about workflows, the workflows execute in an unpredictable way. In the traditional BPEL process, all the web services may be in different service containers, so it is not feasible to preform global resource scheduling. When multiple Hadoop workflows concurrently execute, all the running jobs of these workflows are in one Hadoop cluster. Therefore, how to efficiently schedule these jobs in Hadoop should be a key feature for the Hadoop workflow engine. In this paper, we propose a method to design a Hadoop job scheduler collaborating with a BPEL engine to implement multiple workflows online scheduling.

The novelty of our work mainly lies in the four aspects.

- Propose an approach to build a powerful workflow system for Hadoop using BPEL, which can express complex data processing process.
- Enable Hadoop workflow to easily integrate with existing business process, and enrich business process management with big data processing capability.
- Present a dynamic scheduling strategy for multiple Hadoop workflow applications, leveraging job dependence information and execution time estimation.
- Propose the Hadoop workflow benchmark, which can generate various workflows with required properties.

Section 2 describes the architecture of Pony and how to map the Hadoop workflow model with BPEL model. Section 3 introduces how to schedule multiple workflows in Hadoop. Section 4 gives evaluation experiments. Section 5 discussed the related work. Section 6 gives conclusions and describes future work.

2 System Overview

The principle of the architecture of Pony is simple without modification of the BPEL engine and Hadoop cluster. The system should be extensible to easily support other Hadoop job types and data processing activities in future. Fig.2 is the systematic overview of Pony. BPEL is a little complex for developers to define Hadoop workflow. Therefore, we have developed a graphical design tool to define the dependencies between jobs and the control logics. The tool helps developer design a workflow visually and translates it to BPEL workflow.

The deployment includes deploying BPEL package to BPEL engine and uploading Hadoop Job Java classes to the server where the service container is on. The Hadoop workflow execution engine uses a BPEL engine to orchestrate the jobs by invoking a wrapper in the service container. Only wrapper services interact with Hadoop clusters. Web service wrappers are intended to be an isolated point for the portion of code that does the following: (1) reserializes the operation and parameter data from the parsed

SOAP request. (2) deploys Hadoop Job Java classes to the Hadoop cluster. (3) calls the Hadoop jobs. (4) seializes the return from the jobs and builds the body portion of the SOAP response. This architecture allows the Hadoop jobs to be deployed as a web service while not requiring changes to handle request data in the SOAP format.

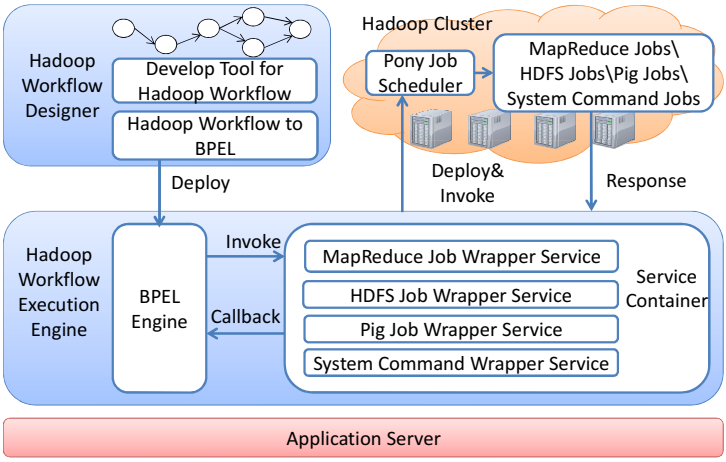

Fig. 2. The Architecture of Pony

3 Mapping Hadoop Workflow with BPEL Process

There are many differences between Hadoop workflow metamodel and BPEL meatmodel. First in the semantic level, two models contain different elements, different semantics for each elements, and different grammatical rules. Second in the deployment and execution level, two models have different activities involved methods, different activities deployment methods and different activities execution methods. Therefore, we must mapping these two model in this two level. Luckily, we find BPEL model can cover the semantic of Hadoop workflow, and we can find the mapping rules between their metamodels.

We first present a method for mapping the metamodel of Hadoop workflow to the metamodel of BPEL. Then, a Hadoop workflow instance can be translated to BPEL instance using the mapping rules. We also describe how to deploy and execute a Hadoop workflow using a BPEL engine.

3.1 Hadoop Workflow Metamodel

A Hadoop workflow is a Directed Acyclic Graph (DAG) of Hadoop jobs and routers. Hadoop workflow can be modeled as HPWFgraph, a graph object. G= (V, E), and V= (V1, V2 , … ,Vn | Vi∈StartNode | EndNode | MapReduce Job | Pig Job | HDFS Job | Cmd Shell | Router | Web service | Java program).

MapReduce job is denoted as MR= {MRName, {Map}, Reduce, {RMap}, Paramerters, Input, Output}. MRName is the task name, Reduce is the reduced function, RMap processes the output to reduce. {Map} denote a set of map functions ,because one MapReduce job may contains more than one kind Map task. Parameters are the setting properties of tasks, Input is the input data source of task, and Output is the output data source of task. The other job's model is similar to MapReduce Job.

E= (e1, e2,…,en) is a set of directed edges. ei= {ID, WFID, Name, Type, Vs, Vt}. ei denotes one edge and represents the dependencies between two nodes. In Hadoop workflow, most dependencies are data dependencies. ID is the identifier of ei, WFID is the identifier of workflow, Name is the name of ei, Vs is the source node and Vt is the target node.

3.2 BPEL Metamodel

BPEL provides a language for the specification of executable and abstract business processes. BPEL are defined by XML lanaguages. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration both within and between businesses.

BPEL2.0 [12] defines the metamodel, and the basic elements include:

- Partners: An important part of the use of BPEL is the description of the business process interaction between partners communicating with Web Services.
- Data manipulation: In BPEL it's possible to use variables. A variable is always connected to a message from a WSDL.
- Basic activities: Every basic activity has several standard attributes and elements that can be used to specify certain properties. In the explanation on this page we will not get into all the options offered by these attributes and elements.
- Structured activities: Structured activities offer a way to structure a BPEL process. They describe the flow of a process by structuring basic activities. In this way control patterns, data flow, fault handling and coordination of messages can be achieved.
- Scope: BPEL uses scopes to split the process up into several parts.

3.3 Translating Hadoop Workflow to BPEL Process

We implemented the automatic transformation between the two models by defining mapping rules. The model transformation engine follows a top-down strategy, and translates a Hadoop workflow instance to BPEL definition files with Java class libraries of jobs. A Hadoop workflow instance is modeled as a HPWFGraph (Hadoop Workflow Graph) object, which contains a StartNode as the start node and a EndNode

as the end node. First, we define a BPEL process object and variable elements to store all the input information of HPWFGraph. Then, we judge the type of the subsequent node of StartNode. If it is a job node, we translate it to an assign and an invoke activity. If it is a router, we translate it to a switch activity or flow activity. Then we continue to translate the subsequent nodes until the end node. Table 1 shows the mapping rules.

Table 1. Mapping Hadoop workflow metamodel and BPEL metamodel

Hadoop Workflow Metamodel Elements		**BPEL Metamodel Elements**
HPWFGraph	Hadoop workflow graph object, with the context information.	Process
		Variable
StartNode/ EndNode	The start node and end node of Hadoop workflow graph.	Sequence
		Receive
		Reply
Job (MapReduce Job/ HDFS Job/Cmd Shell/ Pig Job/Java Application/ Web Service)	Job nodes, includes: MapReduce job, Pig job, HDFS job, System command, Java application, web service and so on.	Assign
		Invoke
If/else	Select the path with the condition	Switch
		Sequence
Fork/join	Parallel split/Parallel merge	Flow
		Sequence

3.4 Hadoop Workflow Deployment and Execution

MapReduce job has a property, that it should be deployed to Hadoop only when the input data are ready. The deploy operation will split the input data and distribute map tasks to salve node. So we should deploy a MapReduce job to Hadoop only when its input data are ready. We present a two-phase deployment strategy. In the first phase, we deploy BPEL definition files and Java classes generated from a Hadoop workflow to BPEL engine. In the second phase, when one job is allowed to run, we use the wrapper service to deploy it to the Hadoop cluster. This deployment strategy does not need any modification of BPEL engine.

With an invoke activity a process can call another Web Service that has been defined as a partner. The invoke activity can be either asynchronous or synchronous. A synchronous invoke needs both an input and an output variable. In an executable business process, these input and output variables are required. An asynchronous invoke only needs to specify an input variable, this because there is no direct reaction and thus no output variable. Because most Hadoop job may need a long execution time, we should use asynchronous invoke. When a Hadoop job finishes, the BPEL engine should be notified to invoke the subsequent job.

4 Efficient Workflow Scheduling

4.1 Workflow Scheduling Problem

Job scheduling in Hadoop is performed by a master node, which splits a job to some tasks and distributes them to a number of slaves. Tasks are assigned in response to heartbeats (status messages) received from the slaves every few seconds. Each slave has a fixed number of map slots and reduce slots for tasks. Slot is the logic resource corresponding to CPU core number. Typically, Hadoop tasks are single-threaded, so there is one slot per CPU core [15]. We need to design special scheduler for workflows. User can submit workflow directly to the scheduler and let the scheduler in chief of executing the jobs as their dependency relation. Workflow scheduling is above the job scheduling layer. Workflow scheduler should be in chief of the slot allocation for multiple workflows to achieve best performance in system perspective. Workflow scheduler should plan how to distribute jobs of workflows to nodes in Hadoop cluster.

When a user submits a workflow application, a key question he or she wants to ask is what the turnaround time will be, which is measured by the time difference between submission and completion of the application. In addition, the makespan is used to measure the workflow application performance, the time difference between the start time and completion time of a workflow. In this paper, we focus on how to schedule multiple MapReduce workflows efficiently at runtime.

4.2 Scheduling Strategy

Some research works are about multiple workflows scheduling in a grid. Their goals are also to minimize average makespan of workflows. The principle is to let the workflow with shortest execution time finish as early as first [15]. This is a NP hard problem. We can use a heuristic algorithm to find a better schedule plan. Many algorithms have been developed since, however most static algorithms are designed in the problem domain of scheduling single workflow applications, thus not applied to a common cluster environment where multiple workflow applications and other independent jobs compete for resources. Workflow applications may be submitted at different times by different users.

Online Workflow Management (OWM) has been proposed for the online mixed-parallel workflows [16]. In OWM, there are four processes: Critical Path Workflow Scheduling (CPWS), Task Scheduling, Task Rearrangement and Adaptive Allocation (AA). CPWS manages the task interdependence and submits tasks into the waiting queue according to the critical path in workflows. The task scheduling process in OWM sorts waiting queue. In the task parallel task scheduling, there may have some slacks among the tasks when the free processor are not enough for the first task in the waiting queue. The multi-processor task rearrangement process works for minimizing the slacks with latter tasks in the queue to improve utilization. When there are free resources, AA takes the highest priority task in the waiting queue, and selects the required resources to execute the task.

In a multi-tenant Hadoop cluster, Pony also needs a online scheduling strategy. We use OWM strategy to design an algorithm for Pony. BPEL engine manages the job dependencies and submit ready jobs to the waiting queue in Hadoop cluster. We design a Hadoop job scheduler to manage the waiting queue. We should sort the jobs in the queue to achieve minimize average makespan of workflow. We will estimate the execution time of each job, and sort them using the execution time ascending. When the free processors are not enough for the first job in the waiting queue, we pick a latter job that can be executed. When there are free resources, AA takes the highest priority job in the waiting queue, and selects the required resources to execute the task. So the key step is how to accurately estimate the execution time of MapReduce job.

Compared to workflow scheduling in grid, it is much harder to estimate the execution time of each job in Hadoop. There have been some related works. This is not the scope of this paper. The user can choose a most suitable algorithm for their application scenario and integrated into the scheduling strategy. We implement the algorithm in Pony

4.3 Scheduler Implementation Architecture

In a Hadoop cluster, there are two approaches to implement the workflow scheduler.(1) Implement the workflow scheduler and deploy it along with the job tracker; (2)Implement a workflow scheduler outside the Hadoop, and implement a job scheduler in Hadoop to cooperate with the workflow scheduler to control the resource allocation. However, we hope to achieve the scheduling of the workflow without any modifications of the Hadoop existing structure. The workflow engine is responsible for the job dependencies of each workflow. The Hadoop job scheduler can schedule jobs from different workflows. The existing scheduler does not know the information of the workflow, and therefore cannot effectively carry out the scheduling on workflow level. We propose a method to make jobtracker know the information about workflow, and then we have developed a job scheduler for Hadoop. Please refer to Figure 2. We only put forward architecture specific scheduling algorithm. User can design their workflow scheduler algorithm and implement a Hadoop job scheduler.

5 Experiments

We have implemented Pony base on OnceBPEL, which is developed by our team one year ago and implements BPEL 2.0. We also implemented the algorithm presented in section 4 to a workflow scheduler for Hadoop. We will compare Pony with Oozie [3], which is the most popular workflow engine for Hadoop. When we use Oozie to execute workflows, we can use three kinds job scheduler in Hadoop: FIFO scheduler, Capacity scheduler, and Fair scheduler. When we use Pony to execute workflows, we use our scheduler.

5.1 Experiments Settings

The experiments are conducted in our in-house 10-node Hadoop cluster. Each node has an Intel (R) Core (TM) i7-2600 CPU @ 3.40GHz, 16GB memory, and two 1TB hard drives, connected to a Gigabit switch. The version 0.20.2 of Hadoop is installed on the cluster. One node serves as the master node and the other as the slave nodes. The single master node runs the JobTracker and the NameNode, while each slave node runs both the TaskTracker and the DataNode. Each slave node is configured with eight Map slots and eight Reduce slots. Each Map/Reduce process uses 500MB memory. The data block size is set to 64 MB.

To evaluate Pony with workflow scheduler, we design a bench for workflow scheduling in Hadoop clusters. As we know, this is the first benchmark for Hadoop workflow. We present a method to randomly generated Hadoop workflows structured according to several workflow graph properties as described in [17], including workflow size, meshing degree, edge length, node- and edge-weight.

To generate workflow graph with different properties, we first randomly generated 1000 workflows by randomly connect two jobs. Then we automatically analyze the workflows and tag them with a suitable class name. For example, one workflow may be tagged with {low size, medium meshing degree, high edge length, high Node-weight and high Edge-weight}. When we evaluate the algorithm, we choose workflows with tags as needed.

To generate various MapReduce jobs in workflows, we design a job template whose input and the output are both texts. The computing logic of the job template is to transform the terms by matching with regular expression. The input data size is between 1GB and 10GB. Therefore, we can use the job template to generate job instances and connect them to create workflows.

Besides the graph properties, we add another set of properties to model the dynamic workload: number of concurrent workflows and arrival interval at which the workflows are submitted into the environment.

5.2 Result and Analysis

The experimental results are analyzed with respect to the evaluation metrics described in the previous section. The metrics include workflow graph characteristics and workload dynamic characteristics of arrival interval and concurrency.

We executed 5, 10, 15 number of concurrent workflows respectively in the experiment. Fig. 3 and Fig. 4 show how the algorithms perform with different number of concurrent workflows. Workflows are randomly chosen from candidates. As a result, Oozie with Fair and Capacity have almost identical performance with respect to average makespan and turnaround. When the number of concurrent workflows is 5, Capacity is better than Fair. Because we set fixed number queues in Capacity Scheduler, and fixed pools in Fair Scheduler, so when jobs number grow bigger, they behavior similarly. Surprisingly, FIFO performs better than Fair and Capacity. Because we compute the average makespan and tour around, so with FIFO, which job is waiting would not influence the result. FIFO only harms the users' experience.

Pony is faster than the other three. Because, the scheduler of Pony aware the information of workflow execution status, and the algorithm helps it make the best decision to minimize the makespan. It can be easily seen that the more concurrent workflows are submitted, the Better Pony outperforms other Oozie with three schedulers.

In the experiments, we assume the arrival interval follows a Poisson distribution with mean value of 0, 5, 15, 20, 25 seconds respectively. We use same 10 workflows in this group experiments and let they arrive one by one. Fig 5 and Fig 6 helps us to understand how the algorithms respond to the workload intensity measured by the interval between workflow submissions. It can be seen four system behavior smoothly with different arrival interval. Because the Hadoop cluster processes jobs quickly and most jobs finished in 5 second. However Pony outperforms other Oozie with three schedulers with different arrival interval of workflow.

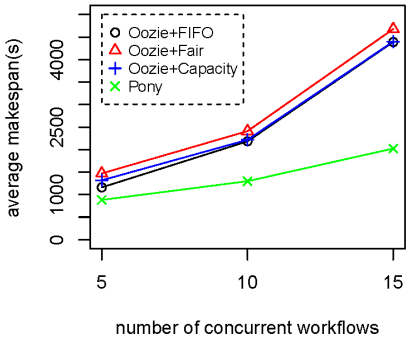

Fig. 3. Average makespan against the total number of concurrent workflows

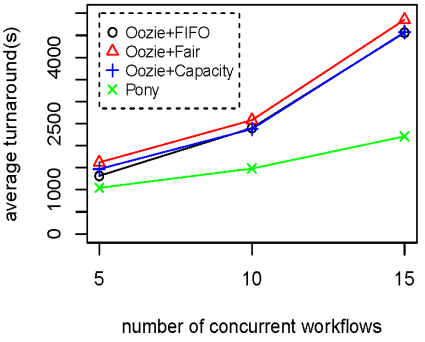

Fig. 4. Average turnaround against the total number of concurrent

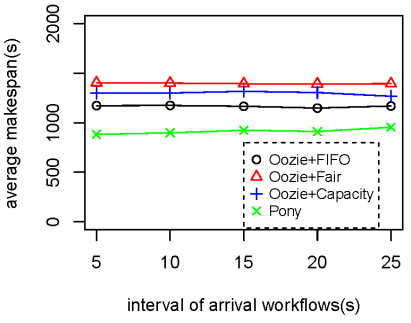

Fig. 5. Average makespan against the arrival interval of workflows

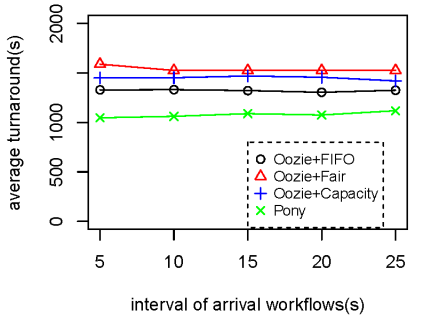

Fig. 6. Average turnaround against the arrival interval of workflows

6 Related Work

There are some Hadoop workflow systems with their own features. Oozie is a coordination system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclic Graphs (DAGs) of actions. Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability [3]. Cascading provides a programming language, using the "pipe and the filters to define the data processing process [4]. Hamake is a lightweight client tool, using the “fold” and “foreach” incremental data processing [5]. Azkaban is still not mature, and its goal is to provide a friendly user interface and time trigger [6]. CloudWF supports both MapReduce workflow and legacy non-MapReduce process workflow [7]. Nova is a pig / Hadoop workflow scheduler to handle real-time data [8]. Oozie is most similar to Pony. However, Pony provides a richer model semantic and efficient enactment engine.

Our Hadoop workflow model is directed acyclic graph, which can be mapped to BPEL freely. [11] proposes a platform-independent conceptual model of ETL processes based on the Business Process Model Notation (BPMN) [10] standard. They also show how such a conceptual model can be implemented using Business Process Execution Language (BPEL). We focus on using BPEL to define Hadoop workflow and execute it. [12] proposed how to define scientific workflow using BPEL. [13] proposes using BPEL to support ETL process execution, and they focus on the data transferring.

Now Hadoop provides three job schedulers. Originally, Hadoop employed First in First out (FIFO) scheduling, but such simple schemes cause job starvation. Capacity Scheduler [14] supports for multiple queues, where a job is submitted to a queue. All jobs submitted to a queue will have access to the capacity allocated to the queue. There are also some research works on job scheduling in MapReduce clusters. Workflow based scheduling has always been a core research area in Grid computing. Yu and Buyya [15] surveyed the research on workflow scheduling in grid computing. [16] proposed Online Workflow Management (OWM) for the online mixed-parallel workflows. [17] present a planner-guided scheduling strategy for multiple workflow applications. Hadoop workflow scheduling is very different from grid workflow scheduling, because the execution time estimation of job is much easier for grid workflow. In this paper, our contribution is introducing multiple workflow schedulers to Hadoop platform.

7 Conclusion

We present a method for quickly building a Hadoop workflow engine based on BPEL: Pony. Pony uses BPEL language to define complex Hadoop workflow, which can integrate with existing enterprise business process. Existing workflow engine does not provide workflow scheduler. We present a workflow level scheduler for Pony to improve dynamic scheduling performance by guiding it with information about the workflow resource demanded estimation. A workload benchmark is developed and the experiment shows its great improvement of performance. We present a flexible architecture for user to design their domain specific scheduler.

Big data makes organizations smarter and more productive by enabling people to harness diverse data types previously unavailable, and to find previously unseen opportunities. Hadoop as a most popular big data processing platform will be adopted by more and more enterprises. We will study how to integrate big data analytical workflow with some real business processes. We also would like to study the feasibility of using BPMN to design a business process with big data processing workflow.

Acknowledgements. This work was partially supported by the National Natural Science Foundation of China (61202065, 61173005), the National Grand Fundamental Research 973 Program of China (2009CB320704), the National Key Technology R&D Program (2012BAH05F02, 2012AA011204).

References

1. Hadoop: Open source implementation of MapReduce, http://lucene.apache.org/hadoop/
2. Jeffrey, D., Sanjay, G.: MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the OSDI 2004, pp. 1–10 (2004)
3. Apache Oozie, http://yahoo.github.com/oozie/
4. Cascading, http://www.cascading.org/
5. Hamake, http://code.google.com/p/hamake/
6. LinkedIn Azkaban, http://sna-projects.com/azkaban/
7. Zhang, C., De Sterck, H.: CloudWF: A computational workflow system for clouds based on Hadoop. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 393–404. Springer, Heidelberg (2009)
8. Olston, C., Chiou, G., Chitnis, L., et al.: Nova: continuous Pig/Hadoop workflows. In: Proceedings of SIGMOD 2011, pp. 1081–1090 (2011)
9. OASIS. Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
10. OMG. Business Process Modeling Notation (BPMN), http://www.omg.org/docs/formal/09-01-03.pdf
11. El Akkaoui, Z., Zimanyi, E.: Defining ETL workflows using BPMN and BPEL. In: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP (DOLAP 2009), pp. 41–48 (2009)
12. Emmerich, W., Butchart, B., Chen, L., et al.: Grid Service Orchestration Using the Business Process Execution Language (BPEL). Journal of Grid Computing 3(3-4), 283–304 (2005)
13. Behnen, M., Jin, Q., Saillet, Y., Srinivasan, S.: Supporting ETL Processing in BPEL-Based Processes. Publication number: US 2008/0115135 A1 (Filing date: November 13, 2006)
14. Capacity Scheduler for Hadoop, http://Hadoop.apache.org/common/docs/current/Capacity_scheduler.html
15. Yu, J., Buyya, R.: Workflow Scheduling Algorithms for Grid Computing, Technical Report, GRIDS-TR-2007-10
16. Hsu, C., Huang, K., Wang, F.: Online scheduling of workflow applications in grid environments. Future Generation Computer Systems 27(6), 860–870 (2011)
17. Zhifeng, Y., Weisong, S.: A Planner-Guided Scheduling Strategy for Multiple Workflow Applications. In: Proceedings of the ICPP - Workshops, pp. 1–8 (2008)