International Journal of Advance Research in Science and Engineering
Volume No.07, Special Issue No.05, March 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

Task Scheduling Algorithm for Data Locality in Cloud Computing

Nitin A. Dhawas[1], Vaishali N. Dhawas[2], Nitin W. Wankhade[3]

Nutan Maharashtra Institute of Technology, Talegaon Dabhade (MS) India[1,3]

Sinhgad Institute of Technology, Lonavala, (MS) India[2]

ABSTRACT

Large scale data processing is increasingly common in Cloud Computing systems like Hadoop, Mapreduce etc. In these systems, files are split into many small blocks and all blocks are replicated over several servers. To process files efficiently, each job is divided into many tasks and each task is allocated to a server to deal with a file block. Because network bandwidth is a scarce resource in these systems. Enhancing task data locality (placing tasks on servers that contain their input blocks) is crucial for the job completion time. Although there have been many approaches on improving data locality, most of them either are greedy and ignore global optimization, or suffer from high computation complexity. To address these problems, we propose a heuristic task scheduling algorithm in which an initial task allocation will be produced at first, and then the job completion time can be reduced gradually by tuning the initial task allocation. By taking a global view, the algorithm can adjust data locality dynamically according to network state and cluster workload.

Keywords—**Cloud Computing, Task Scheduling, Data Locality, Hadoop, Sector**

I.INTRODUCTION

Cloud computing is Internet-connected mode of supercomputing. It is a new type of shared infrastructure, which puts the huge system pool together by the way of operators and the customer.Cloud computing is designed to provide on demand resources or services over the Internet, usually at the scale and with the reliability level of a data center. Clouds provide a large pool of resources, including high power computing platforms, data centers, storages, and software services. It also provides management to these resources such that users can access them ubiquitously and without incurring performance problems.

Balance-Reduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. It is a style of parallel programming that is supported by some capacity-on-demand-style clouds such as Google's BigTable, Hadoop, and Sector.

In this paper, a load-balancing algorithm that follows the approach of the dynamic task allocation by balancing the load on cluster node and then reduces the cost of performance, which increases the efficiency of the system

Fig.1 shows the detailed architecture of the complete system, platforms, software. (Ubantu Operating System, Hadoop database and Sun Java 6 for the platforms, the Java language for programming and HTML, JSP and

XML as the scripting language are used for implementation).This cloud architecture has both a master and slave nodes.

In this architecture a main server is maintained that gets clients requests and handles them depending on the type of request. Search request from the client are forwarded to the node controller which takes care of the tasks for balance and reduce processes. Once the balance-reduce function of the task is completed, the node controller returns the output value to the server and in turn to client.

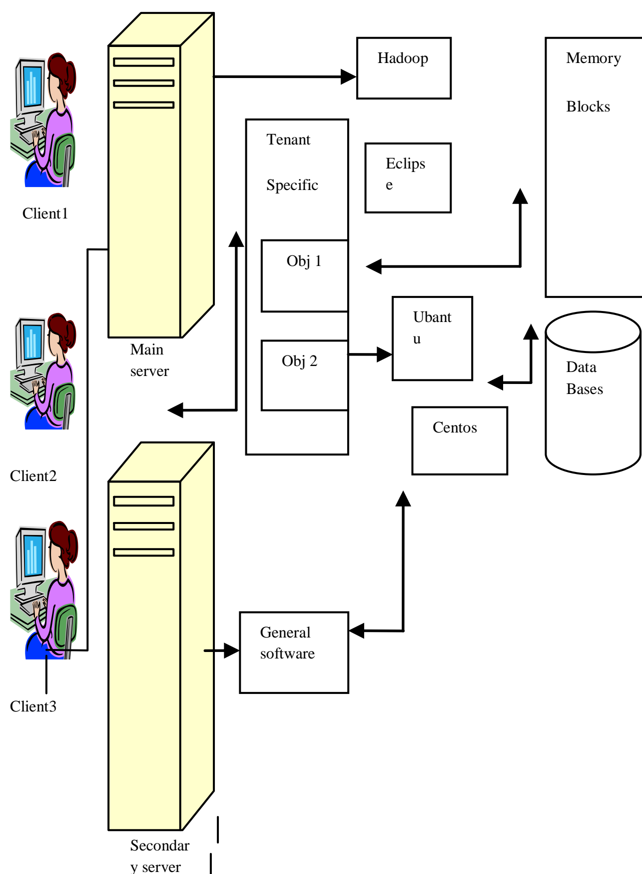

Fig 1:-Architecture of Cloud computing

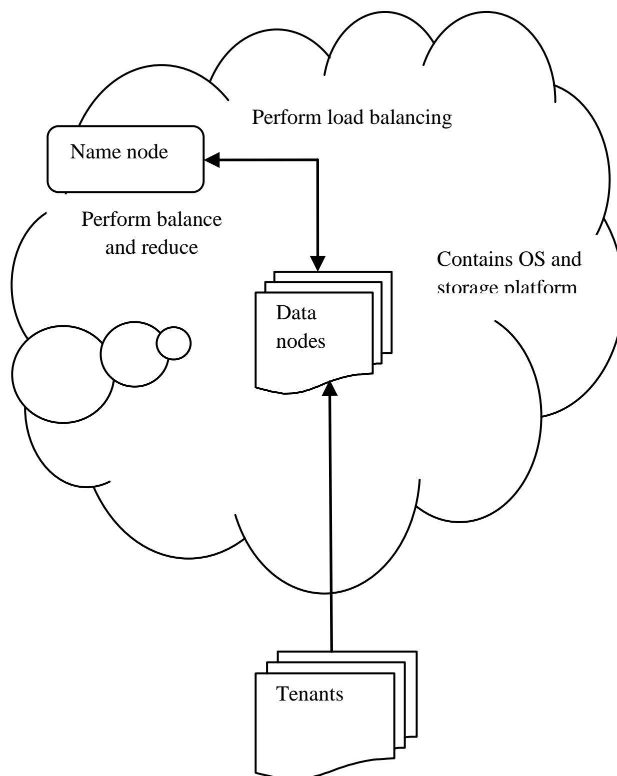

Fig 2. Flow Diagram of System

Fig.2 shows the flow of the system. Here we provide the multitenancy feature of SaaS, in which a single instance of the software serves a number of tenants. So for the same set of software images, there will be different instances generated based on the tenant id

II. RELATED WORK

Over the recent past, a considerable body of work has been done on the use of task scheduling systems for scientific applications. Some of them have investigated scheduling technology with respect to our target applications and scheduling management. Following are the some of the existing scheduling algorithms used in cloud computing.

A. Data-aware scheduling on distributed systems

Over the past decade, data-intensive applications are emerged as an important part of distributed computing. Meanwhile considerable work has been done on data-aware scheduling on distributed systems. Stork [11] is a specialized scheduler for data placement and data movement in Grid. The main idea of Stork is to map data close to computational resources. Though Stork can be coupled with a computational task scheduler, no attempt is made to use data locality to reduce data transfer cost. The Gfarm [12] architecture is designed for petascale

data-intensive computing. Their model specifically targets applications where the data primarily consist of a set of records or objects which are analyzed independently. In Gfarm, several greedy scheduling algorithms are implemented to improve data locality. However these algorithms do not take account of the global optimization of all tasks. Raicu et al. [4] have implemented task diffusion on Falkon [10]. Data diffusion acquires compute and storage resources dynamically, replicates data in response to demand, and schedules computations close to data. Its task scheduling policy sets a threshold on the minimum processor utilization to adjust data locality and resource utilization. However, the simple policy can not improve system performance significantly.

B. Scheduling on cloud computing systems

Scheduling on cloud computing systems has been studied extensively in early literature. The default Hadoop scheduler schedules jobs by FIFO where jobs are scheduled sequentially. To achieve data locality, for each idle server, the scheduler greedily searches for a data-local task in the head-of-line job and allocates it to the server [12]. However the simple policy leads to limited data locality; meanwhile the completion time of small jobs is increased. To enhance both fairness and data locality of jobs in a shared cluster, Zaharia *et al.*[2] propose delay scheduling which improves max-min fairness, when the job that should be scheduled next according to fairness cannot launch a data-local task, it waits for a small amount of time, letting other jobs launch task instead. As servers are assumed to become idle quickly enough, it is worth waiting, for a local task. However, this assumption is too strict, so delay scheduling does not work well when servers free up slowly. A close work to delay scheduling is Quincy [5]. Quincy maps the scheduling problem to a graph data structure according to a global cost model, and solves the problem by a well-known min-cost flow algorithm. Quincy can achieve better fairness, but it has a negligible effect on improving data locality. Hadoop on Demand (HOD) is a management system for provisioning virtual Hadoop clusters over a large physical cluster. It is inefficient that map tasks need read input splits across two virtual clusters frequently. To reduce the data transferring overhead in HOD, Seo *et al*. [6] designs a prefetching scheme and a preshuffling scheme. However, these methods occupy much network bandwidth, so system performance may be decreased. To optimize the performance of multiple Map-Reduce workflows, Sandholm *et al*. [7] develop a dynamic prioritization algorithm, but data locality is not enhanced in this algorithm. To discover task straggler, Zaharia *et al*. [9] propose a system called LATE that makes better estimates of tasks' rest execution time. It is shown that LATE executes speculative tasks more efficiently than the Hadoop's current scheduler in heterogeneous environments where the performance of servers are uncontrollable. To assign tasks efficiently in Hadoop, Fischer *et al*. [3] introduced an idealized Hadoop model called Hadoop Task Assignment problem. Given a placement of input blocks over servers, the objective of this problem is to find the assignment which minimized the job completion time. It is indicated that Hadoop Task Assignment problem is NP-complete. To solve the problem, a flow-based algorithm called MaxCover-BalAssign is provided. MaxCover-BalAssign works iteratively to produce a sequence of assignments and output the best one. It computes in time $O(m^2n)$, where m is task number and n is server number. The solution has been shown to be near optimal. However, it takes a long time to deal with a large problem instance

International Journal of Advance Research in Science and Engineering
Volume No.07, Special Issue No.05, March 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

III. PROBLEM FORMALIZATION

Here we use the task scheduling algorithm which is used first for the balancing the load on cluster and then reduce the performance cost. We consider scheduling a set of independent tasks on a homogeneous platform. On one hand, as input blocks are fixed-size, we assume that data-local tasks take identical constant local cost. On the other hand, as a larger remote task number will cause a higher network contention, remote cost is increased when the remote task number become larger. A job is not completed until all tasks are finished. In addition, we take account of cluster workload: at the start time, if most servers are idle, the cluster is under loaded; in an overloaded cluster, many servers can not be idle in a short time. Base on these assumptions, our goal is to find an allocation strategy that minimizes the job completion time. Load Balancing is used to make sure that none of our existing resources are idle while others are being utilized. To balance load distribution, we can migrate the load from the source nodes which have the surplus workload to the comparatively lightly loaded destination nodes. While balancing load we take care of data, we schedule the task on the server where the data required for that server is present.

As shown in Fig. 3, there are m ($m = 7$) tasks and n ($n = 3$) servers, where each task processes an input block on a server. On one hand, as input blocks are fixed-size, we assume that data-local tasks take identical constant local cost. On the other hand, as a larger remote task number will cause a higher network contention, remote cost is increased when the remote task number become larger. A job is not completed until all tasks are finished. In addition, we take account of cluster workload: at the start time, if most servers are idle, the cluster is under loaded; in an overloaded cluster, many servers can not be idle in a short time. Base on these assumptions, our goal is to find an allocation strategy that minimizes the job completion time. This problem has been shown to be NP-complete in a restrict case (all servers are idle at the start time).

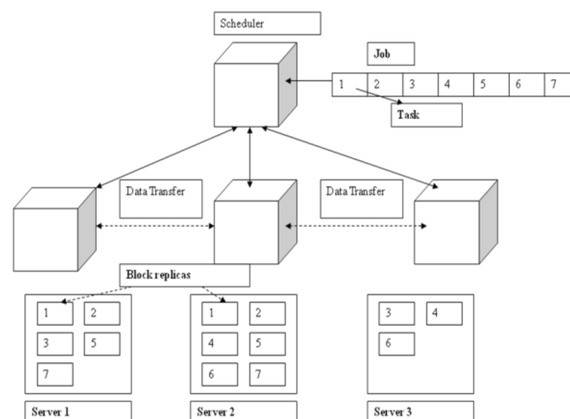

Fig 3: Task Scheduling Process

International Journal of Advance Research in Science and Engineering
Volume No.07, Special Issue No.05, March 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

IV. BALANCE-REDUCE ALGORITHM

In this section, we introduce a data locality driven task scheduling algorithm, called Balance-Reduce. On finding a feasible solution, a critical obstacle is that the remote cost can not be calculated before the remote task number is known. Moreover, it is hard to obtain a near-optimal solution when the remote cost changes frequently. For example, when we allocate a remote task, the remote task number increase by one, so the remote cost may also increase. Furthermore, the load of the servers which have been allocated remote tasks must be updated.

In order to make sure the remote cost, algorithm is split into two phases, *balance* and *reduces*:

• ***Balance***: Given a data placement graph *G*, initial load set *Linit* and a local cost *Cloc*, the balance phase returns a total allocation *B*. Under *B*, all tasks are allocated to their preferred servers evenly.

• ***Reduce***: Given a local cost *Cloc*, a remote cost function *Crem*(·), a total allocation *B* computed by the balance phase, and an initial load set *Linit*, the reduce phase works iteratively to produce a sequence of total allocations and returns the best one. By taking advantage of *B*, the remote cost can be computed at the beginning of each iteration.

Algorithm 1 Balanced Allocation

Procedure BALANCE (G (T Ù S, E), Cloc, Linit)

Define: Gf is a flow network. N is the set of nodes in Gf Gr is the residual graph. T is the iteration number. Pt is an augmenting path at iteration T. TreeT is the search tree at iteration T. Ft is the flow on Gf after iteration T. B is a total allocation.

Gf ← GetFlowNetwork(G)

Gr ← GetResidualGraph(Gf)

N ← T Ù S Ù {Nt}

S.Num ← 0

T ← 1

While T ≤ m do

N.Visited = False

While P = null do

S0 ← MinLoadUnvisitedServer (S, Cloc, Linit)

(Pt, TreeT) ← Augment (S0, Nt, Gr)

N.Visited =Ttrue

Clear (TreeT)

If P = Not Null then

S0.num ← S0.num + 1

End if

End while

S0.num ← S0.num + 1

End while

Ft ← UpdateFlow (Ft −1, Pt)

Gr ← UpdateGraph (Gf, Ft)

T ← T + 1

End while

B ← FlowToAllocation(Fm)

Return B

End Procedure

Algorithm 2 Reduce Makespan

Procedure REDUCE (Cloc, Crem, B, Linit)

Define: P is a remote task pool, Lp is a local partial allocation, R and Rpre are total allocations, Mexp is an expected makespan.

P ← ϕ

Lp ← B

Rpre ← B

While true do

Smax ← MaxLoadActiveServer(Lp)

Tl ← RandomTask (Smax, Lp)

P ← P {Tl}

Lp ← RemoveTask(Lp, Tl)

Mexp ← MaxLoad(Lp)

Crem ← Crem(|P|)

R ← AllocateTasks (P, Lp, Crem, Mexp, Linit)

If makespanR > Mexp then

If makespanR ≥ makespanRpre then

Return Rpre

Else

Return R

End if

End if

Rpre ← R

End while

End Procedure

Algorithm 3 Balance-Reduce Algorithm

Combining the balance phase and the reduce phase, the pseudocode of Balance-Reduce is shown in Algorithm 3.

Procedure BALANCE-REDUCE (G,Cloc, Crem, Linit)
Define: B, R are total allocations.
B ← Balance (G, Cloc, Linit)
R ← Reduce (Cloc,Crem, B,Linit)
Return R
End Procedure

V.HARDWARE AND SOFTWARE REQUIREMENT

To deploy a minimal cloud infrastructure we will need at least two dedicated systems.

1. Front end
2. One or more nodes

For Front End

On one system we install

The Cloud Controller (clc)

The Cluster Controller (cc)

Walrus (the S3-like storage service)

The Storage Controller (Sc)

Hardware	Minimum	Suggested
CPU	1GHz	2*2GHz
Memory	2GB	4GB
Disk	5400 rpm IDE	7200rpm SATA
Disk	5400 rpm IDE	7200rpm SATA
Disk space	40GB	200GB
Networking	100Mbps	1000Mbps

The other system is nodes, which will run as Node Controller

Hardware	Minimum	Suggested
CPU	VT extensions	VT,64-bit, multi core
Memory	12GB	4GB

Disk	5400 rpm IDE	7200rpm SATA or SCSI
Disk space	40GB	100GB
Networking	100Mbps	1000Mbps

VI. PERFORMANCE EVALUATION

In this section, we present several simulations in order to investigate the effectiveness of our algorithm. For comparison, four related task scheduling algorithms are listed as follows:

• MaxCover-BalAssign (MB) [3]. This algorithm works iteratively to produce a sequence of total allocations, and then outputs the best one. Each iteration consists of two phase *maxcover* and *balassign*. Since the remote cost is unknown, it calculates the *virtual cost* which is a prediction of the remote cost. Then it computes a total allocation by taking advantage of the virtual cost.

• Hadoop Default Scheduler (HDS) [8]. When a server is idle, the scheduler chooses a data-local task, then allocates

the task to the server. If there is no feasible task, then the scheduler will select a random task.

• Delay Scheduling (DS) [2]. It sets a delay threshold. If a server is idle and there is no task prefers the server, the scheduler skips the server and increases the delay counter by one. However, if the delay counter exceeds the delay threshold, the scheduler allocates a remote task to the server and sets the delay counter to be zero.

. Good Cache Compute (GCC) [4]. This policy is similar to DS. It sets a utility threshold which is the upper bound of the number of idle servers. The scheduler can skip servers when the idle server number is below the utility threshold. In our simulations, the utility threshold is set to TotalServerNumber $\times$ 90%.

We evaluate the computation time of our algorithm. Since HDS, GCC and DS are run-time scheduling algorithms, we implement them in a compile-time scheme. Firstly, we place all servers into a min-heap; secondly, pop a minimum load server, invoke a real-time scheduling algorithm to allocate a task, then update load of servers. The remote cost is renewed when a remote task is allocated. We do the second step repeatedly until all tasks are allocated. All algorithms are implemented carefully to reduce the redundant work.

The simulations will be implemented in Java and runs on a HP PC with Intel(R) Core(TM) Quad CPU Q8200 and 4GB memory. The server number is set to 2000, and the task number ranges from 100 to 12800. Fig. 4 shows that when the job scale is small, all algorithms can be finished in one second; however, when the task number exceeds 800, the running time of MB increases significantly. We see that the running time of BAR is slightly longer than the greedy algorithms. When the task number is 12800, the running time of BAR, HDS, GCC, DS0.15 and DS0.25 are 9.1s, 5.5s, 5.4s, 7.1s and 7.3s, respectively, while MB takes 203 seconds. Thus, BAR can deal with a large problem instance in a few seconds.

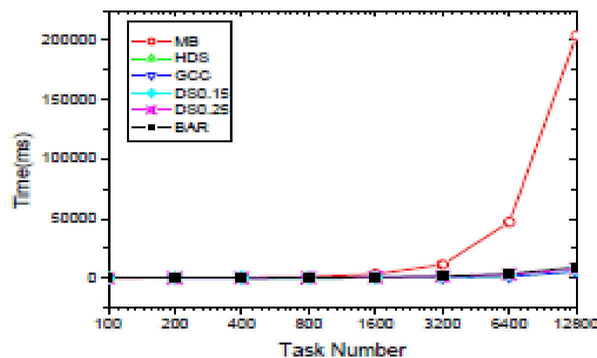

Fig 4. Expected Computation Time

VII. CONCLUSION AND FUTURE WORK

As large scale data-intensive applications grow in popularity, many Cloud Computing systems like MapReduce, Hadoop and Dryad have emerged in recent years. In the general Cloud Computing architecture, network bandwidth is a scarce resource.

To improve the system performance, a task scheduling algorithm must take into account task data locality. However, when most of the servers can not be idle soon and network state is good, enforcing high locality has a negative effect on job completion time.

In this paper, we present a data locality driven task scheduling algorithm called Balance-Reduce. This algorithm schedules tasks by taking a global view and adjusts task data locality dynamically according to network state and cluster workload.

In a poor network environment, this algorithm tries its best to enhance data locality. When cluster is overloaded, this algorithm decreases data locality to make tasks start early. We evaluate this algorithm by comparing it to other related algorithms. The simulation results show that this algorithm exhibit an improvement and can deal with a large problem instance in a few seconds.

As a future work, we plan to implement this algorithm into a production cloud computing system such as Hadoop. In a real world platform, the network state and the cluster workload change frequently, so it is necessary to update the scheduling strategy by an efficient rescheduling algorithm. The rescheduling algorithm is expected to handle machine failure, and network anomaly. However, as the scheduler calls rescheduling frequently, the rescheduling algorithm should be low-complexity.

REFERENCES

[1] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1163–1171.

[2] M. Zaharia *et al.*, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys '10*. New York, NY, USA: ACM, 2010.

[3] M. J. Fischer *et al.*, “Assigning tasks for efficiency in Hadoop: extended abstract,” in *SPAA ’10*. New York, NY, USA: ACM, 2010, pp. 30–39.

[4] I. Raicu *et al.*, “The quest for scalable support of data intensive workloads in distributed systems,” ser. HPDC ’09. New York, NY, USA: ACM, 2009, pp. 207–216.

[5] M. Isard *et al.*, “Quincy: fair scheduling for distributed computing clusters,” ser. SOSP ’09. New York, NY, USA: ACM, 2009, pp. 261–276.

[6] S. Seo *et al.*, “HPMR: Prefetching and pre-shuffling in shared mapreduce computation environment,” in *CLUSTER ’09*. IEEE, 2009, pp. 1–8.

[7] T. Sandholm and K. Lai, “Mapreduce optimization using regulated dynamic prioritization,” in *SIGMETRICS ’09*. New York, NY, USA: ACM, 2009, pp. 299–310.

[8] T. White, *Hadoop: The Definitive Guide*, 1st ed. O’Reilly Media, Inc., 2009.

[9] M. Zaharia *et al.*, “Improving mapreduce performance in heterogeneous environments,” in *OSDI’08*. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42.

[10] I. Raicu *et al.*, “Falkon: a fast and light-weight task execution framework,” ser. SC ’07. New York, NY, USA: ACM, 2007, pp. 43:1–43:12.

[11] T. Kosar and M. Livny, “Stork: Making data placement a first class citizen in the grid,” ser. ICDCS ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 342–349.

[12] O. Tatebe *et al.*, “Grid datafarm architecture for petascale data intensive computing,” ser. CCGRID ’02. Washington, DC, USA: IEEE Computer Society, 2002.

[13] Hadoop. [Online]. Available: http://hadoop.apache.org/