De-De Dodging Algorithm for Scheduling Multiple Workflows in Hybrid Cloud

B.Arunkumar
Assistant Professor
Dept. of CSE
Coimbatore, Tamilnadu, India

T.Ravichandran
Principal
Hindustan Institute of Technology
Coimbatore, Tamilnadu, India

ABSTRACT
Workflow-based applications usually consist of multiple instances depending on a single workflow, which are jobs with control or data dependencies to provide a well-defined scientific computation task, with each instances acting on its own input data. Due to the raise in convention of many applications currently, there is necessitating for high processing and storage capacity along with the consideration of cost and instance use and also without any deadlocks between those instances. To improve the performance of the entire system a high degree of concurrency is obtained by running multiple instances at the same time. On the other hand, since the amount of storage is limited on most systems, deadlock due to numerous storage requests would-be a problem. In this paper we have proposed a new dependency and deadlock avoidance (De-De algorithm) algorithm along with the consideration of both instance and value. The TCHC algorithm that comes to the decision of desiring which resource should be chartered from public providers is now combined with the newly proposed De-De algorithm considering that each instance of both single and multiple workflows should work without any deadlocks. To address this problem, we have combined two new concepts with the traditional problem of deadlock avoidance by proposing a single algorithm that can maximize active (not just allocated) resource utilization and minimize makespan. Our approach is based on the well-known banker's algorithm, but our algorithms make the important distinction between active and passive resources, which is not a part of previous approaches. Through simulation-based studies, we show how our proposed algorithms are better than the classic banker's algorithm.

Keywords
TCHC algorithm, De-De algorithm, Scheduling, Multiple workflows, hybrid cloud.

1. INTRODUCTION
Several high-performance computing (HPC) and a set of computations to be completed, such as those already discussed in bioinformatics [2], [3], biomedical informatics [4], cheminformatics [5] and geoinformatics [6], are complicated workflows of single job. [9] Batch workloads that are typical runs on controlled local area cluster environments. On the other hand organizations that have high workload demands increasingly need ways to share resources across the wide-area, both to lower costs and to increase productivity. One approach to accessing resources across the wide-area is to simply run a local area batch system across multiple clusters that are spread over the wide-area and to use a distributed file system as a backplane for data access. Alas, this approach is loaded with difficulty, largely due to the way in which I/O is handled. The principal problem in using a traditional distributed file system is in its approach to *control*: many decisions concerning caching, consistency, and fault tolerance are made *implicitly* within the file system. Although these decisions are reasonable for the workloads for which these file systems were designed, they are ill-suited for a wide-area batch computing system.

The workflow is usually organized as a *directed acyclic graph* (DAG), in which the constituent jobs (i.e., nodes) are either control or data dependent (i.e., edges).*Control-flow dependency* specifies that one job has to be completed before other jobs start their process. In contrast, *dataflow dependency* specifies that a job cannot start until all its input data (typically created by previously completed jobs) is available [7]. Control-flow is the more commonly used abstraction to reason about the relationship between different jobs, but we show how dataflow information is more valuable to *effectively* utilize the storage. A workflow-based workload may consist of multiple instances of a workflow. Typically, each instance of the workflow is data-independent of other instances since they compute with different inputs or parameters. [8]Additionally, workflows are collaboratively designed, assembled, validated, and analyzed. Workflows can be shared in the same manner that data collections and compute resources are shared today among communities. The scale of the analysis and thus of the workflows often necessitates that substantial computational and data resources be used to generate the required results. [9] So as a remedy for this, Cloud computing is designed such a way that provides on-demand resources to the users, so as to provide locally available computational power, delivering new computing resources when necessary.

Over the last several years, virtual machines have become a usual deployment object. Virtualization advance enhances flexibility because it abstracts the hardware to the point where software stacks can be deployed and redeployed without being tied to a specific physical server. Virtualization technology enables a dynamic datacenter where servers provide a pool of resources that are attached as needed, and where the relationship of applications to compute, storage, and network resources changes dynamically in order to meet both workload and business demands.

With application deployment decoupled from server deployment, applications can be deployed and scaled rapidly, without having to first procure physical servers. Virtual machines have become the prevalent abstraction — and unit of deployment — because they are the least-common denominator interface between service providers and developers. Using virtual machines as deployment objects is sufficient for 80 percent of usage, and it helps to satisfy the need to rapidly deploy and scale applications. Virtual appliances, virtual machines that include software that is partially or fully configured to perform a specific task such as a Web or database server, further enhance the ability to create and deploy applications rapidly. The combination of virtual machines and appliances as standard deployment objects is one of the key features of cloud computing.

Cloud Computing vendors combine virtualization (one computer hosting several "virtual" servers), automated provisioning (servers have software installed automatically) and Internet connectivity technologies to provide the service. Consequentially, acquisition costs are low but tenants never own the technology asset and might face challenges if they need to "move" or end the service for any reason. Something that is often overlooked when evaluating Cloud Computing costs is the continued need to provide LAN services that are robust enough to support the Cloud solution. These costs are not always small. For example, if you have 6 or more workstation computers, you will probably need to continue to maintain a server in a domain controller role (to ensure name resolution), at least one switch (to connect all of the computers to each other and the router), one or more networked printers, and the router for the Internet connection.

Basically these are the following types of the Cloud Services: SaaS (Software as a Service) It provides all the functions of a sophisticated traditional application to many customers and often thousands of users, but through a Web browser, not a "locally-installed" application. It eliminates customer worries about application servers, storage, application development and related, common concerns of IT. Highest-profile examples are Yahoo and Google, and VoIP from Vonage and Skype. PaaS (Platform as a Service) Delivers virtualized servers on which customers can run existing applications or develop new ones without having to worry about maintaining the operating systems, server hardware, load balancing or computing capacity. These vendors provide APIs or development platforms to create and run applications in the cloud – e.g. using the Internet. IaaS (Infrastructure as a Service) delivers utility computing capability, typically as raw virtual servers, on demand that customers configure and manage. IaaS is designed to augment or replace the functions of an entire data center. This saves cost (time and expense) of capital equipment deployment but does not reduce cost of configuration, integration or management and these tasks must be performed remotely. Apart from these we have the following Cloud computing infrastructure models: Public clouds are run by third parties, and applications from different customers are likely to be mixed together on the cloud's servers, storage systems, and networks. Private clouds are built for the exclusive use of one client, providing the utmost control over data, security, and quality of service. Hybrid clouds combine both public and private cloud models. They can help to provide on-demand, externally provisioned scale. The ability to augment a private cloud with the resources of a public cloud can be used to maintain service levels in the face of rapid workload fluctuations. Sometimes called "surge computing," a public cloud can be used to perform periodic tasks that can be deployed easily on a public cloud.

2. DE-DE ALGORITHM DESCRIPTION

Workflows F: {f1, f2, f3…fn}
Deadline E
Resource H
Predestined Start Value PSV
Predestined Finish Value PFV
Public resource pool FB
Private Resource Pool G
Rescheduling group N
Priority Pr
Pending task PT
Application Remaining Time ART
Node set NS
Time & Cost value TCV
Job J with the instance i
Instance of workflows to be scheduled, I_i
Time taken for completion of a job, time ( )
Temporary variables W_i and R_i
Storage request for the job getWriteSet ()
Storage allocation of the job getReadSet ( )
Need of i resources in time t alloc (i, t) / need (i, t)
System safety check safetycheck ( )
Deadlock Dependency Detection Algorithm (De-De),

2.1. ALGORITHM

1) F= Set of Workflows{ F=Workflows==set of tasks TS==single task T}
2) function De-De (I_i , F)
3) $R_i \leftarrow$ getReadSet ( );
4) $J \leftarrow J - (|W_i| - |R_i|)$;
5) alloc (i, t) $\leftarrow$ alloc (i, t) + $(|W_i| - |R_i|)$;
6) need (i, t) $\leftarrow$ need (i, t) − $|W_i|$
7) if (safetycheck (I_i))
8) $J \leftarrow J - |R_i|$;
9) alloc (i, t) $\leftarrow$ alloc(i, t) + $|R_i|$;
10) return true;
11) goto line 19;
12) else
13) $J \leftarrow J + (|W_i| - |R_i|)$;
14) alloc (i, t) $\leftarrow$ alloc(i, t) − $(|W_i| - |R_i|)$;
15) need (i, t) $\leftarrow$ need(i, t) + $|W_i|$;
16) return false;
17) goto line 54
18) End function
19) Perform initial schedule
20) Dependency De=0-5
21) For each W in TW
22) For each T in TS do
23) If T < De Do
24) If (H Є G) then
25) Schedule F in G
26) While (time(F) > E && iteration =F) do
27) Select node from NS with ↑Pr
28) If ni Ɔ NS then
29) Add ni to NS
30) Iteration=iteration+1
31) End while
32) Schedule the H with ↓ PFV
33) De-De (I_i, H);
34) else select next task from TS
35) else select next workflow from WT
36) Else
37) $W_i \leftarrow$ getWriteSet ( );
38) While ($|W_i| > G$ && iteration =F) do
39) Request for H in FB
40) If PFV > ART then
41) Queue PT to execute
42) For each W in TW
43) For each T in TS do
44) If T < De Do
45) Select H Є FB then
46) Calculate TCV for new H
47) If TCV < (H Є G) then
48) Add H to FB
49) else select next task from TS
50) else select next workflow from WT
51) Schedule H with ↓ PFV
52) De-De (I_i, H);

53) End while
54) End else

A cloud system receives numerous numbers of requests for a set of resource to complete their jobs. These jobs are termed as workflows. Each of these workflows consists of set of task which in turn is dependent on one another by some means. In this paper the De-De algorithm consider a set of workflows and detects whether deadlocks occur between them by using the well known banker's algorithm.

The First line of the algorithm initializes the set of workflows that consists of set of tasks T to a variable F. The Function De-De algorithm is defined clearly which includes some of the parameters associated with the instance Ii (i.e., r (t), alloc (i, t) and need (i, t)) are updated accordingly.

In the third line the function De-De is clearly given where Ri is assigned with the allocated resources of the workflows. In the variable G the remaining resource is calculated by subtracting the available resource in the private pool along with the already allocated and requested resources. De-De algorithm first checks if the current available storage is sufficient to satisfy the request of the job (obtained via *getWriteSet ()*). If not, the job has to request from the public resource pool. In line seven the safety check algorithm is invoked for verifying whether the system is in safe state or not for each of the workflow. Once verified the line 19 is called if it returns true. In the 19th line initial scheduling is done in which it considers only the Private resource pool and schedule these workflow in the Private resource pool itself based on some attributes like communication cost, priority and time, resource allocation is done. We have assigned a range for dependency for instance: dependency De value is between 0 - 5. The 23rd line checks the range and once if the dependency value is less than the range, the allocation or request to the resource is done else it is not. Next the algorithm checks whether the available resources are enough or not. If it is sufficient enough to finish the job, the workflow is requested in the private cloud itself else it is requested in the public cloud. Once scheduled the workflows in the private cloud, until the deadline is met the task is running inside the private cloud. The iteration is repeated until the deadline E is met, where the algorithm continues by selecting a node Ni from the node set NS with the highest priority. Then the safety check is algorithm is called.

If it returns true then the system is in safe state else system is said to wait and next workflow is considered.

Simultaneously if the resource is not enough in the private pool it is requested in public pool as in line 39. The line 40 in the algorithm verifies whether the Predestined Finish value PFV is greater than ART, then queue the tasks to execute. Again the dependency range is checked for the new and once if the dependency value is less than the range, the allocation or request to the resource is done else it is not. In line 46 evaluate the new TCV for new resource allocation. Once the value of TCV is less than the available resource in the private cloud then only the public cloud is requested. Since the TCV is considered to be less than the old TCV the resource is added to the set NS. Now schedule the resource with the lowest PFV, suppose the TCV value is larger than verify inside the private cloud itself. In line 41 the De-De algorithm is invoked again for checking safety and if it returns true allocate the resource with the lowest PFV. Finally our algorithm is well furnished to bind between selecting public and private cloud and allocates the requested resources to the particular workflow with the low cost and time and without any occurrence of deadlocks and dependencies between them successfully.

3. RELATED WORK

Deadlock is one of the most discussed problems in the field of operating systems. The theoretical background of this problem as well as its resolution methods have been ingrained and widely deployed since decades ago. As divergent to the traditional batch-oriented workflows, data streaming workflows are continuous and long running in nature, requiring efficient and everlasting transmission of data. The deadlock resolution is particularly vital in these HPC applications because they require high storage cloud be potentially overwhelmed by the incoming data stream if the data arrival rates over take the processing rates but are not properly controlled. Zhang *et al.* [10], [11], has studied this problem and premeditated a suite of repertory strategies to control the start and finish times of the data transfers by setting up upper and lower storage limits. Their storage-aware strategies are based on admission control, a variant of deadlock prevention practice, which is different from ours. As such the recent results in this area are few and far between.

However, in this paper we have provided a case study to show how this problem can be effectively addressed in computational multiple workflows by extending the traditional methods with exploitation of the workflow features. The De-De algorithm attempts to keep the system in safe states, and continues by the use of TCHC algorithm**,** the scheduling process is done by considering both instance and value; effectively provide the selection of choosing between the public and private cloud. Also our paper has included the Banker's algorithm () *a prior* knowledge of the maximum amount of resources needed by each process. Some research efforts focused on refining the banker's algorithm based on some interesting process models, each differing in the amount of information that is assumed to be available [1]. Yu-Kwong Kwok (2004) has made a pair-wise comparison among seven scheduling algorithm under various conditions. But the drawback of this algorithm is that it has a set of several procedures that takes too much time to compile.

A hybrid heuristic scheduling algorithm was implemented on heterogeneous system that comprised of three phases (Sakellariou (2004)). The key idea of the hybrid heuristic is to use a standard list scheduling approach to rank the nodes of the DAG and then use this ranking to assign tasks to groups of tasks that can be subsequently scheduled independently. Rahman, M., (2007). Haluk Topcuoglu has provided two performance-effective and low complexity task scheduling algorithms namely HEFT and CPOP algorithms for heterogeneous system. Edwin.S.H.Hou has developed a genetic algorithm for multiprocessor scheduling Hou, (1994). The algorithm is based on the precedence relations between the tasks in the task graph. He has compared the genetic algorithm with the list scheduling and optimal schedule using random task graphs and a robot inverse dynamics computational task graphs for various are presented. But this existing algorithm does not provide an optimal solution to the scheme.

4. CASE STUDY

A thorough study has been made using the simulator; it depicts the clear view about scheduling the multiple workflows. Initial scheduling is carried out using TCHC algorithm, TCHC determines the scheduling of prioritized task to the public cloud. Once the task is scheduled to carry out the computation, it is checked against the storage allocation.

The de-de dodging algorithm shows significant result when comparing with other algorithm; we assume that maximum claim for a task is 400 units, because maximum claim has to be predetermined in banker's algorithm. The storage from a non workflow based instance cannot be determined with the release

of allocated storage. The banker's algorithm is outperformed by the DDA algorithm, where they have a dynamic storage allocated for different instances. The instances scheduled to the public cloud are chosen in such a way that it has less inter dependency; the requested resources are buffered in local resource pool. Figure 1 shows the dynamic storage and calculating desired resource dynamically allows the de-de dodging algorithm to show a significant result.

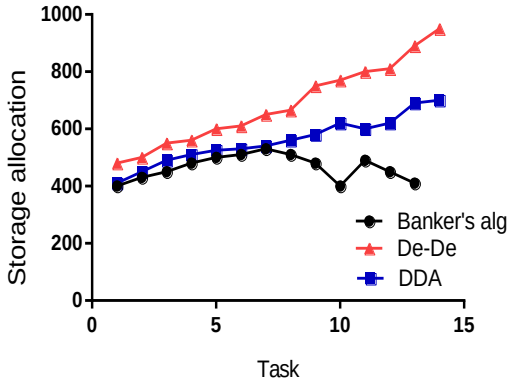

Fig1: Resource utilization

Bankers algorithm, computes the task with maximum available storage and other task needed to execute has to wait, until the executing task finishes the job. The waiting time for the task increases rapidly, if the inter dependency among the task is high. The Figure 2 shows that DDA algorithm has two states active and inactive, which makes it efficient in terms of allocating and deallocating the required space on demand, but it cannot manage the inter dependency concurrency. The proposed de-de algorithm uses TCHC algorithm to set priority to the task, which can run on public cloud.

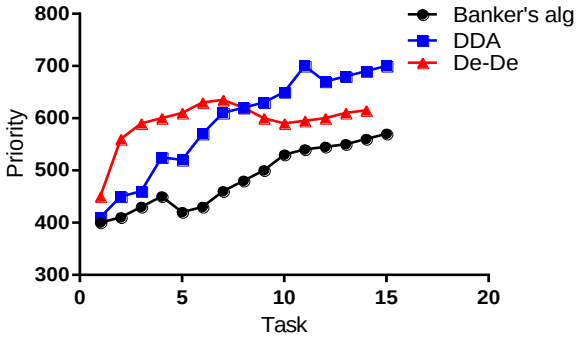

Fig2: Priority

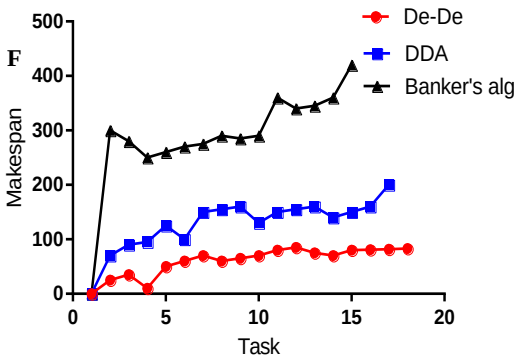

Fig3: Makespan

The Figure 3 shows the number of workflow against the deadline (makespan). Each task has to execute within the desired makespan. The proposed de-de dodging algorithm curve shows that increase in number of multiple workloads will not affect in computing within the desired makespan.

So by the study made on comparing these algorithms we conclude that our proposed De-De algorithm has improved utilization on the available resource and better finds a suitable resource selection between public and private clouds.

5. CONCLUSION

Nowadays in many organizations the needs of extra resources are prevailing in a massive range and as a solution to this is the hybrid clouds, which are being used to execute different kinds of applications. Among them, workflows have an important role in processes of many fundamental science fields, such as Physics, Chemistry, Biology, and Computer Science. To speedup science advancements, it is important to provide efficient resource utilization and to execute the service without any deadlocks among them is the major task nowadays. So as a remedy for this in this paper we have designed a De-De algorithm to speed up the execution of multiple workflows obeying a desired execution time and running the system in a safe state by the use of De-De algorithm which is providing us with a better utilization compared to the DDA and TCHC approach.

The far-reaching estimation carried out in this work provides sufficient data to support the conclusion that the De-De algorithm can provide efficient scheduling in a hybrid cloud scenario and also maintaining the system in a safe state. Its multicore awareness, along with the cost and time knowledge, can provide makespans as low as the user needs. In general, the proposed algorithm has the ability of reducing the execution costs and time in the public cloud with the increase of the workflow desired execution time. Finally conclude by providing that the De-De method has the potential to achieve better resource utilization because information on the "localized approximate maximum claims" is used for testing system safety by the use of banker's algorithm.

7. REFERNCES

[1] Yang Wang and Paul Lu, "Maximizing Active Storage Resources with Deadlock Avoidance in Workflow Based Computations" Ieee Transactions On Computers-2012

[2] T. Werner, "Target gene identification from expression array data by promoter analysis," *Biomolecular Engineering*, vol. 17, pp. 87–94, 2001.

[3] D. Szafron, P. Lu, R. Greiner, D. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, and D.

Meeuwis, "Proteome analyst: Custom predictions with explanations in a webbased tool for high-throughput proteome annotations," *Nucleic Acids Research*, vol. 32, pp. W365–W371, 7 2004, http://webdocs. cs.ualberta.ca/~bioinfo/PA/.

[4] GROMACS, http://www.gromacs.org.

[5] M. Schmidt, K. Baldridge, J. Boatz, S. Elbert,M. Gordon, J. Jensen, S. Koseki, N. Matsunaga, and J. Montgomery, "The general atomic and molecular electronic structure system," *Journal of Computational Chemistry*, vol. 14, pp. 1347–1363, 1993,http://www.msg. ameslab.gov/GAMESS/GAMESS.html.

[6] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, 2005.

[7] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528– 540, May 2009.

[8] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Mayers, and M. Samidi, "Scheduling data-intensive workflows onto storage-constrained distributed resources," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 401–409.

[9] J. Bent, D. Thain, A. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny, "Explicit control in a batch-aware distributed file system," in *Proceedings of Networked Systems Design and Implementation (NSDI)*, San Francisco, California, USA, 2004, pp. 365–378.

[10] W. Zhang, J. Cao, Y. Zhong, L. Liu, and C. Wu, "An integrated resource management and scheduling system for grid data streaming applications," in *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, ser. GRID '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 258–265.

[11]"Block-based concurrent and storage-aware data streaming for grid applications with lots of small files," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 538–543.