2016 IEEE 36th International Conference on Distributed Computing Systems

# On MinMax-Memory Claims for Scientific Workflows in the In-Memory Cloud Computing

Yang Wang[†], Chengzhong Xu[†§], Shuibing He[‡], and Xian-He Sun[¶]
[†]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, China
[‡]School of Computer, Wuhan University, China
[§]Department of Electrical and Computer Engineering, Wayne State University, USA
[¶]Department of Computer Science, Illinois Institute of Technology, USA
{yang.wang1,cz.xu}@siat.ac.cn, heshuibing@whu.edu.cn, sun@iit.edu

***Abstract*—We propose a new concept of minmax memory claim (MMC) to achieve cost-effective workflow computations in in-memory cloud computing environments. The minmax-memory claim is defined as the minimum amount of memory required to finish the workflow without compromising its maximum concurrency. With MMC, the workflow tenants can achieve the best performance via the maximum concurrency while minimizing the cost to use the memory resources. In this paper, we present the algorithms to find the MMC for workflow computation and evaluate its value by applying it to deadlock avoidance algorithms.**

## I. Introduction

A scientific workflow generally consists of a set of data-dependent tasks, forming a *weighted directed acyclic graph* (DAG), also called *workflow graph*, to carry out a complex computational process. The nodes in the workflow graph represent tasks (e.g., executable or a script) that accomplish a certain amount of work in the workload, and edges denote the data channels used to transfer data volume from source node to target node. In a cloud-based workflow computation, the data channels are typically implemented via an external provisioned storage system (e.g., a file system), which could incur substantial disk I/O overhead that can dominate the execution times.

To address this issue, in-memory caching utility in the cloud provides an effective way, which aggregates massive memory resources across a dedicated cluster of servers to support all the data managements via a middleware software [1], [2]. With in-memory caching, the workflow computation could transfer data between tasks via fast, managed, in-memory caches, instead of relying entirely on slower disk-based file systems. Compared with disk read/write operations, this enhancement could come with potentially several orders of magnitude better end-to-end latency, and thus substantially improve the overall performance of the workloads.

However, given the intrinsic complexity of the workflows, it would be very hard, if not impossible, for the workflow tenants to make accurate reservations on the resources to be used. Oversubscription or undersubscription would result in either unproductive spending or performance degradation.

To address this problem, we propose a concept of *Minmax-Memory Claim* (MMC) in this paper for the workflow computation, which is defined as the minimum required memory resources without compromising the workflow concurrency (also the performance). The MMC is desirable for cost-effective computing in the cloud because the amount of memory provisioned over the MMC claim cannot make any performance contribution. As such, it is very beneficial to those who intend to have maximized performance while minimizing the budget for the provisioned resources in the cloud.

## II. Algorithms for MinMax-Memory Claim

### A. Basic Ideas

The basic idea of the proposed algorithms is first to augment the workflow graph with an *edge-node transformation* and compute the *Maximum Weighted Concurrent Set* (MWCS) of the *augmented workflow graph*, then provably show the MWCS is the MMC of the original workflow graph. Since the MMC is not a fixed value, rather, it is monotonically decreased as more tasks in the workflow are finished, the weight of the MMC can be viewed as the maximum claims of the remaining tasks in the instance.

### B. MinMax Memory Claim

*1) Basic Definition:* At any time instance $t$ during the computation, we can classify the nodes in the graph into three groups:

- *Done(t)*: the nodes that have been completed prior to $t$.
- *Running(t)*: the nodes in $V - Done(t)$ that are running concurrently at $t$
- *Blocked(t)*: the remaining nodes at $t$, i.e., $V - (Done(t) \cup Running(t))$.

Accordingly, we can define the amount of memory resources that held in each set of nodes as follows,

$$\varphi_{d\to r}(t) = \sum_{v_i \in Done(t), v_j \in Running(t)} w(e_{ij}) \quad (1)$$

$\varphi_{d\to r}(t)$ is the total memory resources that are created by $Done(t)$ and being used by $Running(t)$ at $t$

$$\varphi_{d\to b}(t) = \sum_{v_i \in Done(t), v_j \in Blocked(t)} w(e_{ij}) \quad (2)$$

Similarly, $\varphi_{d\to b}(t)$ is the total memory resources that are created by $Done(t)$ and will be used by $Blocked(t)$ at $t$.

$$\varphi_{r\to b}(t) = \sum_{v_i \in Running(t), v_j \in Blocked(t)} w(e_{ij}) \quad (3)$$

1063-6927/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDCS.2016.13

Finally, $\varphi_{r\to b}(t)$ is the total memory resources that are created by $Running(t)$ and will be used by $Blocked(t)$ at $t$. As a consequence, the *Minmax Resource Claim* (MMC) at time $t$ is determined by the maximum of $\varphi_{d\to r}(t)+\varphi_{d\to b}(t)+\varphi_{r\to b}(t)$, which is also the minimum memory resources to ensure the maximum concurrency of the workflow. Thus, we define the MMC of the workflow graph $G(V,E)$ as

$$MMC(G)=\max_{t\in[0,l]}\{\varphi_{d\to r}(t)+\varphi_{d\to b}(t)+\varphi_{r\to b}(t)\} \tag{4}$$

where $l$ represents the makespan of the workflow instance.

*2) Our Solutions:* The basic idea of our approach is similar to the one that finds the MWCS in the workflow graph, but with an extension to count the weights of those edges that span across the frontier of the concurrent task set (e.g., $\varphi_{d\to b}$) as well. To this end, we first augment the workflow graph via an edge-node transformation, and then find its MWCS.

Given a workflow graph $G$, we define an *augmented workflow graph* $G_a$ with the following *edge-node transformations:*

- for each node $v$ in the graph, define $w(v)=b_v$;
- for each edge $e$ in the graph, add a dummy node $v_e$ on the edge with a weight defined as $w(v_e)=w(e)$, and in the meanwhile, clear the edge weight (i.e., the augmented workflow graph is only node weighted).

The essence of this transformation is to convert finding *MWCS* with respect to nodes to finding it with respect to edges by treating each edge as a weighted dummy node in the augmented graph. With the augmented workflow graph, we have the following theorem,

**Theorem II.1.** *The weight of the MWCS of the augmented workflow graph is equal to the MMC of the original workflow graph, i.e.,* $\mathscr{W}(MWCS(G_a))=MMC(G)$.

*Proof:* The proof is straightforward as for the weighted concurrent set $WCS(t)$ at time instance $t$ in $G_a$, its occupied memory resources must belong to some active tasks which are running concurrently (i.e., $\varphi_{d\to r}(t)+\varphi_{r\to b}(t)$) or certain memory channels that have been created for later uses (i.e., $\varphi_{d\to b}(t)$). As such, we have

$$\mathscr{W}(WCS(t))\le\varphi_{d\to r}(t)+\varphi_{d\to b}(t)+\varphi_{r\to b}(t)$$

By applying the similar logic, the reverse argument is also correct, i.e., the size of occupied memory resources by concurrent running tasks and memory channels at $t$ is not greater than the weight of $WCS(t)$.

Overall, there is one-to-one mapping between the occupied memory resources of the original workflow graph and the weight of concurrent set in the augmented graph, i.e.,

$$\mathscr{W}(WCS(t))=\varphi_{d\to r}(t)+\varphi_{d\to b}(t)+\varphi_{r\to b}(t) \tag{5}$$

whereby we can conclude the theorem by maximizing both sides of Eq.(5) in terms of $t$ over the workflow makespan. ■

The main ingredient of our algorithm is to compute the *MWCS* on $G_a$, which is equivalent to finding the *Maximum Weighted Clique* (MWC) in a graph derived from $G_a$. As such, it can be solved by some existing algorithms in either optimal or approximiate ways [3], [4].

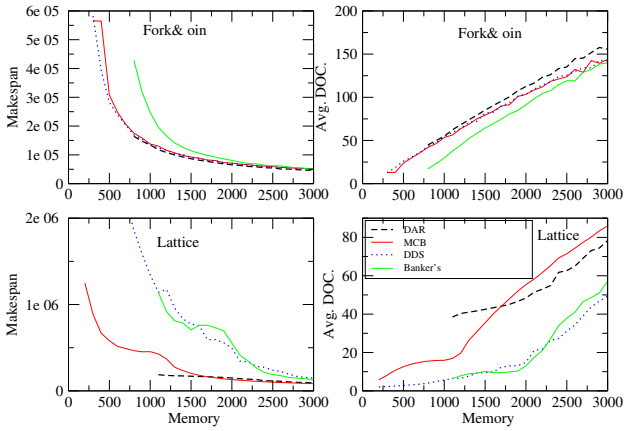


Fig. 1. Makespan comparison with reference algorithms when the memory channel sizes are uniformly varied in [1,10], and job times are uniformly distributed in [500,1000].

## III. Performance Studies

We evaluate the value of MMC by designing a MMC-based deadlock avoidance algorithm (MCB), which is an extension of the banker's algorithm with using MMC as the localized maximum claims. The MMC algorithm is then compared with other two deadlock resolution algorithms, *DAR*[5] and *DDS* [6], for two types of often-used workflow workloads, *fork&join* and *lattice*, each being composed of 100 instances.

Figure 1 shows the results of the *makespan* comparison where we can observe that for the fork&join, compared with *DAR*, *MCB* can work using much less memory resources, which even has the same performance behavior with *DDS*, while in lattice, *MCB*, compared with other algorithms, can tolerate limited memory for the computation at cost of degraded performance since its maximum claim for each instance is much less than that of *DAR*. These results demonstrate the performance gains of using the concept of MMC, compared with the other reference algorithms.

## References


[1] G. Chockler, G. Laden, and Y. Vigfusson, "Data caching as a cloud service," in *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*, ser. LADIS '10. New York, NY, USA: ACM, 2010, pp. 18–21.

[2] ——, "Design and implementation of caching services in the cloud," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 9:1–9:11, Nov 2011.

[3] D. R. Wood, "An algorithm for finding a maximum clique in a graph," *Operations Research Letters*, vol. 21, no. 5, pp. 211 – 217, 1997.

[4] P. R. Östergård, "A fast algorithm for the maximum clique problem," *Discrete Applied Mathematics*, vol. 120, no. 13, pp. 197 – 207, 2002, special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

[5] Y. Wang and P. Lu, "Maximizing active storage resources with deadlock avoidance in workflow-based computations," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2210–2223, 2013.

[6] ——, "DDS: A deadlock detection-based scheduling algorithm for workflow computations in HPC systems with storage constraints," *Parallel Comput.*, vol. 39, no. 8, pp. 291–305, August 2013.