Workflow System Performance Improving and Analyzing by Using Memory Cache

Yusheng Xu, Hongxin An, Qunce Xu
School of Information Science and Engineering
Lanzhou University, Lanzhou
Gansu Province, China
Emails: xuyusheng@lzu.edu.cn, an.hx@163.com, xuqc12@lzu.edu.cn

***Abstract*—Workflow techniques may be broadly used in business processing and scientific computing and thus focused research interest over the last decade. The performance of workflow applications mainly depend upon task scheduling strategies in workflow engine. In this paper, a novel active workflow instance memory cache (WfIC) architecture is proposed, which is used for improving workflow system performance. By caching active workflow instances in main memory, instance database IO operations are pruned during task scheduling and processing. Additionally, theory analysis shows that WfIC can effectively improve the performance of workflow systems. Also, theory analysis gives an instruction on how to configure the instance cache and the lower/upper limit of the instance executing time.**

Keywords—workflow; workflow optimization; workflow performance; workflow active instance cache; workflow overhead; business workflow; scientific workflow

I. INTRODUCTION

A workflow consists of a sequence of concatenated tasks, each of which follows the precedent without gap and ends just before the subsequent ones may begin. These workflow tasks are loosely coupled, and the relationship among them can be modelled as control flow or data dependence flow. No matter which case, a module according to a workflow task can be developed independently, and the total development and maintaining cost can be largely reduced. As to the literatures, more and more systems are deployed in workflow architecture. The basic categories of workflow applications include business processing and scientific computing. Business processing are the primary application systems for enterprises, which produce millions of data records every day and host a large number of business transactions. These systems are mainly developed with navigation-driven models and designed based on the business of one department. Workflow-based enterprise applications have many benefits such as removal of data dependency, removal of flow dependency, flexibility, reusability, integration, scalability etc. Many enterprise applications have been built on workflow environment such as order processing, department coordination, etc. Due to the loosely interdependent characteristic of workflow, large-scale scientific computing in different fields such as astronomy [2], biology [12, 14], earthquake-science [3] is constructed with workflow technology in cluster [7], grid [22] or cloud [4] environment. Workflow has become a key technique in both enterprise business processing and scientific computing domains.

The task of a workflow may be just a simple process, interaction such as recording data into the database where the runtimes are from few seconds to few minutes, or may be as complex as a scientific computing which can continue for several days. In the most complex situation, the task may also be a sub-workflow which contains thousands of sub-tasks. Thus, how to get a high performance workflow system is a big challenge for workflow applications. The performance may be evaluated in several ways such as total runtime of each workflow instance (also known as the makespan), throughput of workflow instance, utility ratio of resource, etc. The runtime is the mostly used metric. The runtime of a workflow instance is defined as the total amount of clock time from the moment the first instance task is triggered until the last task complete. The runtime is composed of overhead and task execution time. As to [19], such overheads may include: unusually large grid latencies, unpredictable waiting times on cluster queues, job status processing time, data staging, and others in scientific computing environment. Fig.1 shows an overheads example, where four tasks of one workflow instance are executed and their runtimes are projected onto x-axis (the gray bars). In this example, total time of overheads is about 60%. Thus overheads have a non-negligible influence on overall workflow performance. There are two directions for performance optimization: to reduce overhead or to improve algorithm of task processing. Task processing algorithm is domain specific and its performance improvement is out of this paper. Reducing overheads inspires us and the active Workflow Instance memory Cache (*WfIC*) technique is proposed.

The contributions of this paper are:

- Proposed a memory cache diagram (*WfIC*) for active workflow instances to reduce instance scheduling and management overheads.
- Made a theory analysis of *WfIC* on reducing overhead.
- Performed experiments in an ideal workflow configuration to compare the effect with and without *WfIC* strategy.

This work is supported by the Fundamental Research Funds for the Central Universities under grant No. lzujbky-2010-92.

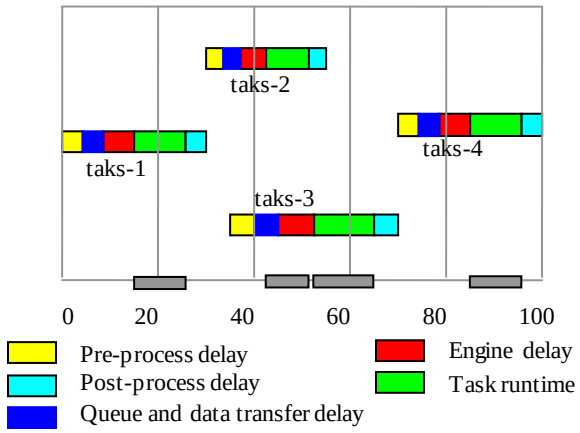

Figure 1. Workflow overheads example

The result of experiments and theory analysis shows that WfIC is an effective strategy for reducing workflow overheads and can be used for both businesses processing and scientific computing, specifically for workflows with large scale of tasks.

The rest of the paper is organized as following. The related works are discussed in section 2. Section 3 presents the details of the active workflow instance cache (WfIC) architecture. A theory analysis on WfIC performance is made in section 4. Finally, the conclusions and future directions can be found in section 5.

II. RELATED WORKS

Over the last decade, workflow technology has gotten great progression. Such model method as SCXML [18], the Petri Nets [8, 15], Business Process Modeling Notation (BPMN) [11], UML Activity Diagrams [10] have been proposed successively. At the same time workflow patterns [16, 21] and soundness verification of workflow model [17] have been matured. All these works just focused on workflow modelling and the performance of workflows is ignored. Based on these techniques, workflow applications are constructed for enterprise and scientific computing in different fields.

Although modelling and soundness verification methods are enough for developing small or middle size workflow systems successfully, there is performance problem when orchestrates large size or complex workflow applications. As a result, cumulative overheads analysis [20] and layered queuing modeling [9] are proposed for performance analysis. In [20], Weiwei Chen et al. presented an overhead analysis method which can be used for performance measurement of scientific workflow applications in distributed environments, such as clouds, grids and dedicated clusters. Additionally, this metric may be used to decide whether an optimization method can reduce some kinds of overhead. [9] describes a layered queuing modelling to investigate the performance of various workflow architectures. Unlike WfIC, these methods make performance analysis of diverse overheads. How to reduce impact of overheads is ignored.

Much research interest is concentrated on workflow performance improvement and such strategies as task clustering [5], task/data throttling [13] and data staging [1] are the mostly used techniques to address the issues of performance optimization. In order to improve performance of fine computational granularity task scientific workflows, task clustering [5] can minimize the completion time of the workflow by reducing the impact of queue waiting time. Using level- or/and label-based task clustering techniques, it can be applied to data intensive applications. In contrast to general methods which achieve high performance by intelligently scheduling tasks on resources, data throttling [13], which proposed a data throttling framework to reduce unbalanced execution/communication and thus got a higher performance scaleup for scientific workflow systems. By staging data in / out, data staging [1] has the potential to significantly improve the performance of scientific workflow applications that have large input data sizes. However, these strategies focus on reducing data transfer upon network and queuing time on resource schedule that they are appropriate for data intensive scientific computing in dedicated cluster, grid or cloud environment. They may not be same effective for enterprise transaction processing workflow.

Hyungjin Ahn et al. [6] proposed message queue-based workflow architecture for large-scale high performance workflow applications. Workcase is defined as a stateful session bean (EJB). This schema, which is made up of a message queue and a workcase pool, is based on the workflow instance-oriented (or workcase-oriented) architectural style and is able to spawn up to 1 million workcases. Unlike message queue, WfIC just manages active instances and tasks with Petri token in memory cache and an instance in just a workflow object.

III. THE ACTIVE WORKFLOW INSTANCE MEMORY CACHE

This section describes the active workflow instance memory cache architecture in detail, which includes the workflow instance object, the cache architecture.

A. *Workflow instance object*

As to Petri Nets model [8, 15], a workflow meta-model is a directed graph consisting of places, transitions and arcs. Places represent possible instance states while transitions describe events that change instance states. And arcs are the relationships between places and transitions. Formally, a workflow meta-model is defined as a 4-tupel Petri Net: $WfN=<P, T, F, M0>$, where $P = \{p_1, p_2, \ldots, p_{|P|}\}$ is a finite set of places, $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is a finite set of transitions and $P \cap T = \Phi$. $F \subseteq P \times T \cup T \times P$ is the set arcs representing relationships between places and transitions, and M_0 is the instance initial status which is defined in the following. Without losing generality, p_1 is the unique input (start) place and $p_{|P|}$ is the exclusive output (end) place of workflow WfN.

A workflow instance represents a complete business case and is processed in temporal order by the workflow application. Marking function $M_t : P \mapsto N$ is the number of tokens (or marks) at temporal time t in a place. The status $Mt(i)$ of a workflow instance i at time t is defined as a vector of marks of places, that is $M_t(i) = [M_{i,t}(p_1), \ldots, M_{i,t}(p_{|P|})]$. Specifically,

$M_0(i)=[1,0,...,0]$ is the initial (or input) status which corresponds to instance i is started or submitted and $M_e(i)=[0,...,0,1]$ is the end (terminate) status which corresponds to the processing of instance i has been terminated. Without confusion they can be denoted as M_0 and M_e respectively.

For simplicity, several assumptions are made. (1) WfN is assumed as a directed acyclic graph (DAG). (2) Place $p_{|P|}$ is just a flag and there is nothing to process for its corresponding task. (3) There is at most 1 mark in a place for each instance. The multiple marks (Petri Net tokens) situation is not considered. (4) All task execution for one place or transition can be terminated in one unified time unit, denoted as t_{pu}, t_{tu} respectively.

If there is an arc $< p,t >\in F$, transition t is called an output transition of place p. The set of all output transitions of place p is denoted as $p\bullet$. A workflow instance i is active if and only if it has been initialized and has not reached the terminated status M_e. For an active instance i in status M_k, its active places is the set of place that has s mark in status M_k and the active place set is denoted as $AP_k=\{p \mid M_{ik}(p)=1 \wedge p\in P\}$. The candidate transition set is the set of transitions that may be triggered next time of instance i in status M_k, and the candidate transition set is denoted as $CT_k=\{t \mid t\in p\bullet \wedge p\in AP_k\}$. All other transitions that are not in CT_k cannot be triggered next time. Here initialization may be submitting or starting in theory. But in reality, submitting may be confused with starting. For semantic clarity in this paper, an instance is submitted means that the instance has been submitted to workflow engine but cannot be scheduled for lack of resources while an instance is started means that the instance is submitted and can be scheduled even though it has not been scheduled.

A workflow instance object is an active instance which is described as instance data, active place set AP_k and candidate transition set CT_k. The instance data is the basic information for workflow scheduling while AP_k is set of active place data and tasks. And CT_k is set of transition tasks.

B. *The active instance cache architecture*

The workflow instance object is frequently accessed and processed during its active lifecycle. If this object is read or/and written from disk when each such task execution as instance scheduling, place and transition do task processing, guard condition checking, authorizing, etc., the performance must be greatly slowdown. On the contrary, if the instance object has already been in the main memory when these tasks are beginning to process, the performance of workflow system may be improved. The active instance cache architecture is such a technique for performance improvement.

The overall active instance architecture is made up of instance object cache, instance database and components of place services and transition tasks. Fig.2 outlines this cache-based architecture. As previously discussed, an active instance object includes three parts: the active instance itself, the active place set and the candidate transition set. The active instance and places may have corresponding data, thus should be stored into memory cache. The instance object cache is composed of index cache, instance cache and place cache. The instance

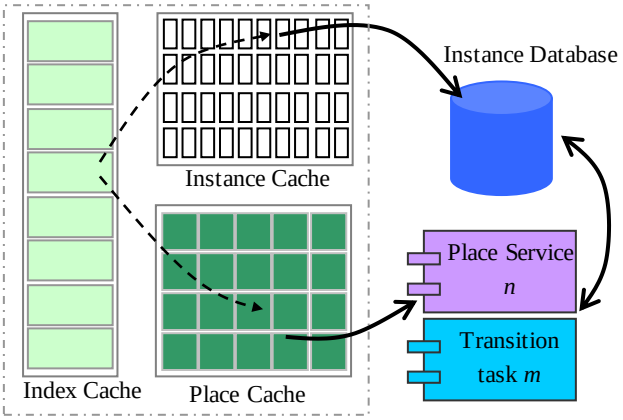

Fig 2. The active instance object cache architecture

cache is a buffer for active instance data about instance state, status, instance scheduling etc. Place cache is for active places of all active instances. These data is used for place processing such as entry, do and exit. To be noted that not all place data is appropriate place cache. Such data as non-structure data in scientific computing is too big for place cache and should be stored in instance database. Index cache is a method for random instance object accessing.

Component of place service encapsulates business logic of places while component of transition task encapsulates logic of transitions which includes firing rules. During the execution, these components access instance database directly if large size of semi or unstructured data is inputted into or outputted from them. Otherwise, they interact with place cache. The instance database component is used to store workflow information permanently and it can be a SQL-based database or a shared data repository, which depends upon the specific workflow application.

C. *Using index cache to reduce memory cache size*

A workflow instance will locate at the active instance cache from it is submitted until it reaches terminate state. At the other side, for reasons of workflow engine schedule strategy, instance type and resource request and allocation method, first submitted instance may be not the first terminated one. That is to say, the duration of each workflow instance lifecycle (the total instance running time) may have major difference. This will be a problem for random instance access. For example, supposed 100 instances are submitted and their *IDs* are from 0 to 99, among which 10 instances have reached the terminate state (their *IDs* are from 1 to 10). For the sake of difference instance lifecycle duration, these 10 instance cache memory cannot be re-allocated and only 90 instance cache units are used while 100 units are occupied by cache.

Under this situation, the index cache is introduced. Instance cache can be random access through index cache without waste instance cache memory. Continue previous example. When 10 instances (ID from 1 to 10) reach terminate state, they are eliminated from the instance cache and their cache memory should be re-allocated to other submitted instances (ID from 100 to 109). But, in index cache, their positions will still exist until instance 0 reaches terminate state. Thus, 110 instance

cache units are reduced to 100 units by using 110 index cache units.

D. Cache operation algorithms

Instance cache in *WfIC* is a specific cache. Instance cache related operations include cache reading, cache writing, instance starting and instance terminating. Reading, writing and replacement are operations for general cache structure. But for *WfIC*, once they are loaded into cache, all active instance objects are always in the cache during their active lifecycle. Thus, each instance processing operation will be hit in the cache and there is no replacement operation occurring during instance reading or writing. These operations can be executed in a straight way.

How to swap the dirty cached data out is a problem for cache structure. In *WfIC* architecture, there are two ways to write dirty data into disk. One way is writing through. Each writing operation will update both cache and disk. Another way is writing log first. Once an update operation occurs in cache, log is firstly written to disk and the dirty data will be written to disk at an appropriate time. Comparing to the first way, writing log first may reduce disk I/O. However, its algorithm is more complex.

As previous discussion, an instance can be scheduled by workflow engine only when it has been started. In *WfIC*, starting an instance means allocating caches (both instance cache and index cache) for it and is a bit more complex. Figure 3 shows the algorithm of starting operation. Line 7 is the pre-condition for starting: there are submitted instance(s) waiting for starting and both index cache and instance cache have free room (not FULL). In line 10, variable n is the number of instances that can be started this time. And then, in line 11-16 these instances are loaded into caches one by one. The starting operation may only be invoked on two occasions when one instance terminates or one instance is submitted and there is no other instance waiting for starting.

When an instance terminates, it must be eliminated from instance cache and the corresponding index cache is set invalid. On this occasion, if index cache has free room then the starting instance operation can be invoked. On the other hand, both instance and index cache may have free units when there is no instance waiting for starting. Thus, when an instance is submitted, it may be started immediately.

When an instance reaches terminate status M_e, the instance processing finishes and should be eliminated from instance cache. These tasks are done by terminating operation, whose algorithm is depicted in Fig.3. The first task is to write instance data into disk and to free the corresponding instance cache (line 19). And then to set invalid flag on the corresponding index cache for reusing in line 20. If the index cache is the first occupied unit, the index cache should be freed (line 21) and then freeing the corresponding cache in line 22-25. In line 26 to invoke starting operation to start some submitted instances.

Fig.4 shows a terminating operation example. In Figure 4(a), the initial status, index cache 00 and 01 are valid and occupied by instance *id1* and *id2* respectively. Index cache 02 is occupied, but it contains an invalid instance. There are four

```
01  Algorithm to terminate an active instance
02  Input: r_list – a list of submitted ready instances but not started
03         idx_cache – the index cache, ist_cache – the instance cache
04         instid – ID of instance to be terminated
05  output: put some ready instances into cache

06  Procedure start( r_list, idx_cache, ist_cache)
07   IF r_list is NULL OR idx_cache is FULL OR ist_cache is FULL
08     RETURN;
09   END IF
10   n = min(r_list.length, idx_cache.freen, ist_cache.freen);
11   FOR (i = 0; i < n; i++)
12    nist = r_list.getremove_instance();
13    nist.ID = idx_cache.newID();
14    idx_cache.put_instance(nist);
15    ist_cache.put_instance(nist);
16   End FOR
17  END start;

18  Procedure terminate(instid, r_list, idx_cache, ist_cache)
19   ist_cache.writeout(instid);
20   idx_cache.set_invalid(instid);
21   IF idx_cache.first_element(instid)
22    WHILE idx_cache.invalid(instid) DO
23      idx_cache.free(instid);
24      instid = idx_cache.nextID(instid);
25    END WHILE;
26    CALL start(r_list, idx_cache, ist_cache);
27   END IF
28   END terminate;
```

Fig3. Instance start algorithm

submitted instances named *A*, *B*, *C* and *D* in the ready queue (*r-list*). Now instance *id2* reaches its terminated state and terminating operation is invoked. The instance cache is freed, but the index cache (unit 01) is just set an invalid flag and is not freed. Fig.4(b) shows the state when this operation finishes. And then, the business processing of instance id1 has finished. This time, both instance cache (*id1*) and index caches (00, 01 and 02) are freed. Index cache 00 is freed because instance *id1* is the first element in index cache. Index cache 01 and 02 are freed because they are invalid when cache 00 is freed and they are neighbors of first elements. This situation is shown in Fig. 4(c). Additionally, as Fig.4(d) depicted, instance starting operation is invoked and instances *A*, *B* and *C* are loaded into cache (*idA*, *idB* and *idC*) from instance ready queue.

IV. PERFORMANCE ANALYSIS OF INSTANCE CACHE

By caching active instance in memory, WfIC may improve performance of workflow applications. This section gives a theory performance analysis of instance cache. The goal is to find which factor impacts performance in the instance cache architecture.

A. Performance improvement

For every instance, each place and transition of workflow $WfN=<P, T, F, M0>$ has at least one input operation and one output operation without instance cache. The input operation reads in place or transition related data before their processing while output operation writes them back. Introducing instance cache, *WfIC* can reduce the number of instance database I/O at a great degree. The number of instance database read is 1 with instance cache. However, without instance, the number is

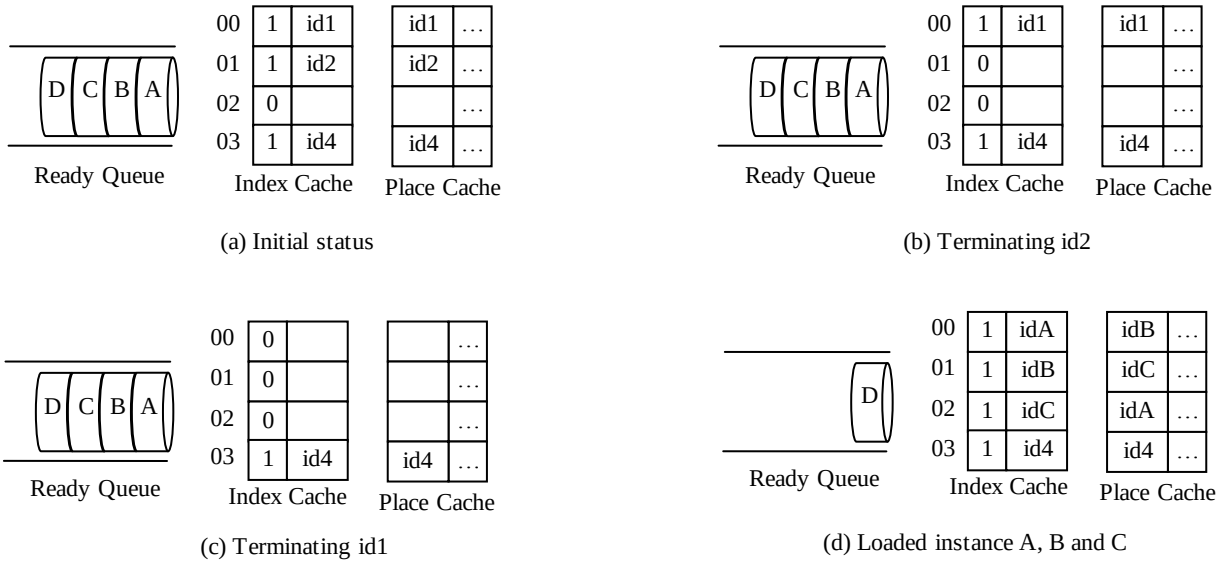

Fig 4. Example of terminating operation

$|P|+|T|$. But for instance database writing, how many can be reduced depends on dirty data updating strategy. For example, there is no reduction under write through strategy.

Comparing to existing workflow architectures, in *WfIC* with instance cache, active instance managing operations such scheduling, guard condition checking, etc. can be executed in cache without instance database querying.

B. Factors that affect performance

Reducing number of instance database I/O can improve performance during active lifecycle for each instance. The total executing time of an instance i $t_e(i)$ includes two parts: ready queue waiting time $t_q(i)$ and active time $t_a(i)$. There are several factors that affect the performance of *WfIC* architecture. Suppose I_1, I_2, I_n, … are instances that occur for specific workflow system.

(1) $N_i(t)$ is the maximum number of concurrent instances to be processed for a period of time t. It is an essential attribute of the target workflow system.

(2) Instance active lifecycle time $t_a(i)$ affects the size of instance cache S_c. Obviously, a longer lifecycle times $t_a(i)$ means that it stays in the cache for a longer period of time. For the same throughout, the cache size S_c should be larger.

(3) Instance cache size S_c is a constant for a given workflow application. It mainly depends on $N_i(t)$ and instance active lifecycle $t_a(i)$. It should be an appropriate value that meets the following goal: there are about S_c instances in the cache at most of time while there should not be more and more instances in the waiting queue.

(4) Instance active lifecycle time difference $\Delta t_k(n)$ is the maximum lifecycle difference of a set of n sequential instances. Formally it can be defined as $\Delta t_k(n) = \max\{t_a(I_j) - t_a(I_i) \mid k \le i < j \le k+n-1\}$.

It can be concluded from above definitions that all instances in the cache can reach their terminating status for a period of time no more than $t_a(I_k) + \Delta t_k(S_c)$. During the same period of time, there may be $N_t(t_a(I_k) + \Delta t_k(S_c))$ instances are submitted. And for the same period they must be started. Thus we have

$$N_i(t_a(I_k) + \Delta t_k(S_c)) \le S_c \qquad (1)$$

This equation gives a clue on how to configure the instance cache. Generally, both $t_a(i)$ and $\Delta t_k(n)$ are all about constant parameters which denoted as t_a and Δt respectively. And the lower limit of the size of instance cache is $N_i(t_a + \Delta t)$. Additionally, if instance cache size is increased to $N_i(t_a(I_k) + \Delta t_k(S_c)) + S_c$, there will be almost no instance waiting. The reason is that the submitted instance can be started immediately. How to configure the instance cache is summarised in equation (2).

$$N_i(t_a + \Delta t) \le S_c \le 2N_i(t_a + \Delta t) \qquad (2)$$

As previous discussion, the maximum waiting time for a submitted instance is $t_a(I_k) + \Delta t_k(S_c)$. Thus we have $t_a \le t_e(I_k) \le 2t_a + \Delta t$ based on equation (3).

$$t_a(I_k) \le t_e(I_k) \le t_a(I_k) + \Delta t_k(S_c) + t_a(I_k) = 2t_a(I_k) + \Delta t_k(S_c) \qquad (3)$$

V. CONCLUSION AND FUTURE DIRECTIONS

Workflow is one of the most popular technologies for both enterprise business processing and scientific computing. The performance of workflow applications is a key problem and focuses research interest over the last decade. In this paper, a novel active workflow instance memory cache (*WfIC*) method is proposed, which is used for improving workflow system performance. By caching active workflow instances in main memory, instance database IO operations are greatly pruned during task scheduling, processing and instance management. Additionally, theory analysis shows that *WfIC* can effectively improve the performance of workflow systems. Also, theory analysis gives an instruction on how to configure the instance cache and the lower/upper limit of the instance executing time.

For future works, the first direction is to integrate *WfIC* with current leading workflow management systems and mature workflow modelling languages and tools. Extending *WfIC* for the grid and cloud computing environment is another direction.

REFERENCES

[1] Ann Chervenak, Ewa Deelman,et al., Data placement for scientific applications in distributed environments, in 2007 8th IEEE/ACM International Conference on Grid Computing, Page(s): 267 - 274, 2007.

[2] D. S. Katz, N. Anagnostou, G. B. Berriman, E. Deelman, et al, Astronomical Image Mosaicking on a Grid: Initial Experiences, in Engineering the Grid: Status and Perspective, American Scientific Publishers, 2006.

[3] Deelman, E., Callaghan, S., Field, E.; Francoeur, H. et al, Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example, in the Second IEEE International Conference on e-Science and Grid Computing, 2006 (e-Science '06), pp.14, Dec. 2006.

[4] Ewa Deelman, Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments, in International Journal of High Performance Computing Applications August 2010 vol. 24 no. 3, PP. 284-298, 2010.

[5] Gurmeet Singh, Mei-Hui Su, Karan Vahi, et al., Workflow Task Clustering for Best Effort Systems with Pegasus, in Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities (MG'08), Article 9 , 8 pages, 2008.

[6] Hyungjin Ahn, Kiwon Lee, Taewook Kim, Haksung Kim, Ilkyeun Ra, Workflow Message Queue's Performance Effect Measurements on an EJB-Based Workflow Management System, in Advances in Web and Network Technologies, and Information Management, Lecture Notes in Computer Science Volume 4537, pp. 467-478, 2007.

[7] Jeffrey S Vetter, Frank Mueller, Communication characteristics of large-scale scientific applications for contemporary cluster architectures, in Journal of Parallel and Distributed Computing Volume 63, Issue 9, September 2003, Pages 853–865.

[8] Khodakaram Salimifard, Mike Wright, Petrinet-based modelling of workflow systems: An overview, European Journal of Operational Research, Volume 134, Issue 3, 1, Pages 664–676, 2001.

[9] Kwang-Hoon Kim and Clarence A. Ellis, Workflow performance and scalability analysis using the layered queuing modeling methodology, in Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work (GROUP '01), pp.135-143, 2001.

[10] Marlon Dumas and Arthur H. M. ter Hofstede, UML Activity Diagrams as a Workflow Specification Language, in Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001), pp. 76-90, Toronto, Canada, 2001.

[11] Object Management Group, Business process modeling notation specification, final adopted specification, February 6, 2006.

[12] Patrick M. Widener, Tahsin Kurc,et al., High Performance Computing Techniques for Scaling Image Analysis Workflows, in Lecture Notes in Computer Science, Volume 7134/2012, pp.67-77, 2012.

[13] Sang-Min Park and Marty Humphrey, Data throttling for data-intensive workflows, IEEE International Symposium on Parallel and Distributed Processing, 2008(IPDPS 2008), Page(s): 1-11, 2008.

[14] Talbot T, Ide D, Liu N and Turchi J, A novel, variable angle guide grid for neuronal activity studies, Front. Integr. Neurosci. 6:1. doi: 10.3389/fnint.2012.00001, 2012.

[15] Van der Aalst, W. and van Hee, K., Workflow Management. Models, Methods, and Systems, the MIT Press, ISBN-10: 0262720469, March 1, 2004.

[16] W.M.P. Van Der Aalst, A.H.M. Ter Hofstede and A.P Barros, Workflow Patterns, in Distributed and Paralled Databases, 14, pp.5-51, 2003.

[17] W.M.P. van der Aalst, Verification of Workflow Nets, in Application and theory of Petri nets, Lecture notes in computer science, vol 1248. Springer-Verlag, Berlin, pp407–426, 1997.

[18] W3C, Working draft, World Wide Web Consortium, 5 July, http://www.w3.org, 2005.

[19] Weiwei Chen, Ewa Deelman, Workflow Overhead Analysis and Optimizations, in Proceedings of the 6th workshop on Workflows in support of large-scale science (WORKS'11), Pages 11-20, 2011.

[20] Weiwei Chen, Ewa Deelman, Workflow Overhead Analysis and Optimizations, in Proceedings of the 6th workshop on Workflows in support of large-scale science (WORKS'11), Pages 11-20, 2011.

[21] Yusheng Xu, Hongxin An, Zhixin Ma and Li Liu, D&B: Two Additional Workflow Patterns, Communications in Computer and Information Science, Volume 201, pp.434-445, 2011.

[22] Zhijie Guan, Francisco Hernandez, et al, Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface, in Concurrency and Computation: Practice and Experience, Volume 18, Issue 10, pages 1115–1140, 25 August 2006.