Distributed Scheduler of Workflows with Deadlines in a P2P Desktop Grid

Javier Celaya and Unai Arronategui
Dept. of Computer Science and Systems Engineering
Universidad de Zaragoza, Zaragoza, Spain
Email: {jcelaya,unai}@unizar.es

***Abstract*—Scheduling large amounts of tasks in distributed computing platforms composed of millions of nodes is a challenging goal, even more in a fully decentralized way and with low overhead. Thus, we propose a new scalable scheduler for task workflows with deadlines following a completely decentralized architecture. It's built upon a tree-based P2P overlay that supports efficient and fast aggregation of resource availability information. Constraints for deadlines and the correct timing of tasks in workflows are guaranteed with a suitable distributed management of availability time intervals of resources. A local scheduler in each node provides its available time intervals to the distributed global scheduler, which summarizes them in the aggregation process. A two phase reservation protocol looks for suitable resources that comply with workflow structure and deadline. Experimental results, from simulations of a system composed of one million nodes, show scalable fast scheduling with low overhead that can allow a high dynamic usage of computational resources.**

I. Introduction

Applications with large workflows and large inter-task concurrency are important in different scientific domains. However, applying time constraints to this kind of applications is still a challenging problem. Traditional computing platforms suffer from many overheads [1] when dealing with this well known problem [2]. Additionally, scheduling such workflows in distributed computing platforms with a large number of nodes implies a new level of complexity. A decentralized solution is needed for scalability, but a good enough schedule must be obtained from inaccurate global state of the system.

We propose a distributed scheduler that is able to manage workflows with deadlines in a computing platform with up to one million nodes. Workflows are scheduled fast and with low overhead, providing users with an easy and simple interface. Our contribution includes a workflow decomposition which allows the global scheduler to match sequences of dependent tasks with time constraints. The scheduler is built upon a tree-based network overlay following a P2P model in the PeerComp computing platform. Availability information of execution nodes is distributed among all nodes of the tree through an aggregation process that maintains this information periodically while smartly bounding network traffic. The global scheduler uses this information in a distributed tree-search operation to allocate workflow tasks, which is bounded to logarithmic time by the balanced nature of the tree.

The scheduler has been evaluated and validated with simulations of a system with one million nodes. Even with this size, only few seconds are required to allocate different kinds of workflows. This allows to work with task lengths of just a few minutes. As far as we know, this is the first scheduler of workflows with deadlines that reaches such scale. In this first approach, we schedule only workflows with negligible data transfers involved. Also, for the sake of simplicity, memory and disk restrictions are not considered, although they can be easily applied as additional constraints similar to deadlines.

The rest of this paper is organized as follows: Section II introduces the related work, and a description of the architecture is provided in Section III. Then, the details of workflow management and scheduling are presented at Sections IV and V, respectively. Finally, Section VI shows the experimental results and Section VII gives our conclusions and future work.

II. Related Work

Traditional grid platforms, as those based in the Condor scheduler [3], show poor scaling with big workflows [2] (up to thousands or hundred thousands of tasks) due to scheduler overheads. Scaling is even worse if most tasks in workflows are of short duration (in the range of minutes). In [1], advance reservations, multi-level scheduling and infrastructure as a service are explored to reduce these overheads. However, high scalability with millions of computational nodes concurrently available that reduces the execution time of big workflows is not considered in these explorations.

A distributed double-layer scheduling model is proposed in [4] for grid workflows. They propose a global scheduler that uses information aggregation so it does not need detailed status information of resources. Also, resource availability fluctuations are considered to allocate the workflows. However, although the two layer approach improves the scalability, no study is offered to analyze it in their model. Their experiments, using 3 resource clusters with 10 nodes each, do not allow to evaluate this feature either. Our approach generalizes some of the features of their proposal with the multi-level approach built in our tree-based P2P model.

In [5], a scheduling algorithm is described based on the cooperation of distributed workflow brokers. A Distributed Hash Table (DHT) provides a P2P coordination space that is responsible for the decentralized resource discovery and

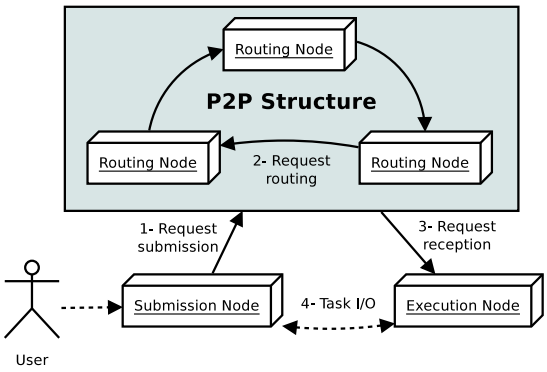

Fig. 1. Node roles. Submission nodes send requests to routing nodes, which forward them to execution nodes. Then, input data and code is transferred from submission node to execution node, and results are given in return.

scheduling. A FCFS allocation strategy is used. Thus, workflows with deadlines can't be scheduled with this algorithm.

III. System Architecture

The work presented in this paper is a scheduling model for PeerComp, which is a fully distributed computing platform over a P2P network for desktop grid applications that extends our previous work [6], [7]. It looks for scalability, simplicity and decentralization properties. Its architecture consists of three main elements: the nodes, the network overlay and the distributed scheduler. In this section, we briefly explain them.

A. Node functionality

Nodes provide the basic functionality of PeerComp, with the capability to cooperate in the distributed algorithms of the scheduling model. This functionality is divided into three roles: the *routing node*, the *execution node* and the *submission node*. Every node plays all three roles, in different positions of the overlay, in order to balance the load of the system management. Fig. 1 shows the interaction between node roles.

The routing node is in duty for maintaining the network overlay structure. Additionally, it aggregates the availability information and routes workflow execution requests to suitable execution nodes. The execution node is the role that actually executes workflow tasks, and is mandatory for all the nodes; that is, every peer has to share its computational resources. Finally, the submission node controls the submission of requests for workflow execution to the network. It provides the user with the interface and the logic to control the tasks and workflows it wants to deliver, and to monitor their progress in the host execution nodes.

B. Overlay Structure Management

To provide a substrate for the distribution of the availability information and the rapid search of multiple nodes at once, a balanced tree-based overlay is used, based on VBI-Tree [8]. This tree is organized so that routing nodes are located in the branches and execution nodes in the leaves. Execution nodes are ordered by IP address, to provide certain locality information to the global scheduler.

There exists certain controversy in the use of a tree-based structure as a scalable network overlay, as the top levels of the tree may turn into its weakness. From the point of view of the structure management, VBI-Tree provides good load balancing and fault tolerance features. However, it is still true that the other parts of the system must also take these problems into consideration, which we comment in the respective sections.

C. Scheduler Architecture

On top of the tree overlay and the nodes lays the distributed scheduler of PeerComp, which presents a two-layer architecture, with a local and global scheduler. Local scheduling defines policies at execution node level, while global scheduling matches workflow tasks with suitable execution nodes. There is a local scheduler instance in each execution node, while the global scheduler is totally decentralized, with its functionality distributed among all the peers of the system.

Local and global scheduler communicate by the former exporting the information about execution node availability to the later, so it can use that information to match workflow requirements and execution node properties. In this paper we will only consider available computing time as the availability information of execution nodes. For a complete solution, other characteristics, like available memory and disk space, could also be taken into account.

The global scheduler provides two main functionalities at each routing node, which assist the matching algorithm. The first one is the availability information management, which aggregates and distributes this information in the tree branches. Later, this information is used in the discovery algorithm, to match requests requirements with execution node availability, and forward requests to suitable subbranches.

IV. Workflow Management

The submission node is in charge of managing workflow delivery. Workflows are modeled after a Direct Acyclic Graph (DAG) $G(V, E)$ which is divided into several parts that can be submitted concurrently, to exploit the inherent parallelism of the graph structure. In such a graph, nodes represent tasks and edges represent dependencies between them. For each $v_i, v_j \in V$, an edge $(v_i, v_j) \in E$ exists if task v_j depends on v_i. In that case, task v_i must finish before v_j may start.

Classic workflow scheduling algorithms [9] usually divide DAGs in levels, so that for every task v_i, if an edge (v_j, v_i) or (v_i, v_j) exists, v_j does not pertain to the same level as v_i. This decomposition is suitable when the objective of the scheduler is to optimize makespan, providing the set of resources where the tasks are going to be allocated in advance. However, PeerComp uses a different approach for DAG decomposition, since PeerComp reserves resources as they are discovered. The DAG is divided into *sequences* of dependent tasks, starting with the longest path and then sequences for which time constraints can be calculated from the sequences they depend on. The longest sequence gets its deadline directly from the DAG, and shorter sequences get their time constraints from the longer ones as they are allocated.

Require: D is a DAG with tasks in $D.V$ and edges in $D.E$.
Ensure: $\mathbb{S}$ contains the sequences extracted from D.
  Extract critical path from D to sequence S.
  $\mathbb{S}.add(S)$
  while $D.V \neq \emptyset$ **do**
    Extract the longest path S from $D.V$ so that:
- $\forall s_i \in S \mid s_i \neq s_1,\ \nexists v_i \in \mathbb{S} \mid (v_i, s_i) \in D.E$
- $\forall s_i \in S \mid s_i \neq s_n,\ \nexists v_i \in \mathbb{S} \mid (s_i, v_i) \in D.E$

  $\mathbb{S}.add(S)$
  end while
  return $\mathbb{S}$

Fig. 2. Algorithm $decomposition(D)$

The decomposition is depicted in algorithm in Fig. 2. It starts by extracting the sequence that contains the critical path of the DAG, in terms of task length. Then, in each iteration, new sequences are constructed by selecting the longest sequence S that fulfills two properties:

1) No edge may go from a task of a sequence already extracted to a task from S, but to the first task.
2) No edge may go from a task of S to a task of a sequence already extracted, but from the last task.

Note that any DAG may be decomposed this way, as at each iteration at least a sequence of one task is eligible. Thus, all nodes are eventually assigned to a sequence.

Once the DAG is decomposed into sequences, they are assigned not only a deadline, but also a *startline*. These two constraints are known after the sequences they depend on are allocated, with the timing information provided by execution nodes. The longest sequence is allocated first, because its deadline is provided by the user and the startline is the current time. After it is allocated, following sequences can be submitted as soon as their time constraints are calculated. At each step, all the prepared sequences may be sent concurrently, as they do not depend of each other. Thus, the submission process consists of a set of stages, in which all the prepared sequences are sent. We define the *width* of a workflow as the number of stages needed to submit the complete workflow. Likewise, we define the *minimum length* of a workflow as the sum of the task lengths in the critical path, and the *total length* of a workflow as the sum of all the task lengths. As we show in the experimental results, these properties have relevant impact in the system performance.

V. Distributed Scheduling

When the global scheduler receives a sequence to allocate, the startline and deadline are used to send it to an execution node with enough available computing time between those limits. In order for the global scheduler to find one or more execution nodes for a sequence, three elements are needed: a local scheduler policy which defines task ordering at the execution node level and exports its availability information to the tree structure, a definition of availability information which includes detailed time information about when nodes will be idle, and a routing algorithm which uses this information to deliver requests to the most suitable nodes.

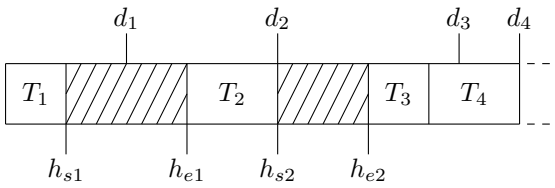

Fig. 3. Task queue with four tasks, and the holes available between them.

A. Local Scheduling Policy

PeerComp uses an EDF-based local scheduler. This policy orders queued tasks by their deadline, with the earliest first. It is known that this is the best ordering to get tasks finished on time. For a sequence of tasks, the deadline of the sequence is applied to the last task, and a deadline is calculated for each of the others from the deadline and the estimated execution time of its successor.

Every time the task queue changes, the availability of an execution node is recalculated, to report when and of which length new sequences could be accepted. To obtain this information, all tasks in the queue are pushed to their deadline, except the currently running task, because task preemption is not allowed[1]. Then, a list is constructed with the "holes" of availability that exist between tasks. These holes represent a simplified view of the maximum length a new sequence may have so that it would make no other task miss its deadline when it is to be accepted by this execution node.

Fig. 3 shows an example of a task queue with four tasks. A hole which starts at h_{si} and ends at h_{ei}, represents the maximum length of a sequence with a deadline in $[d_i, d_{i+1})$, so that in EDF order it would execute before T_{i+1}. Actually, if T_2 is executed just after T_1, the hole between T_2 and T_3 would be longer, but it is not considered for sake of simplicity in the global dispatcher algorithm. Note that there is no hole between T_3 and T_4, as T_4 should have already started by d_3.

The computational power of the node is also reported, so that it can be used to estimate how much work is performed by the node in an arbitrary period of time, like when a node is completely idle.

B. Availability Information

Once the set of holes is created, it is sent to the father routing node so that it will be aggregated with the sets of the other execution nodes in that branch. The aggregation process consists in summing up the number of holes with equal time limits and availability, to produce an *availability summary* of what can be found in a certain branch. However, aggregating sets of arbitrary holes would be impossible. For this reason, a special data structure is designed.

An availability summary structure consists of a set of *reference points*. These points represent moments in time, and

[1]This is the rule in other distributed computing platforms too, so that only one task is using the resources reserved for the platform at any time.

TABLE I
EXAMPLE OF THREE REFERENCE POINT SETS WITH THEIR CREATION TIMES.

	Approximate difference with creation time									
t_c	5min	10min	15min	30min	1hr	2hr	4hr	8hr	16hr	1day
4:33	4:40	4:50	5:00	5:30	6:00	8:00	12:00	16:00	1d:00:00	2d:00:00
5:48	5:55	6:00	6:15	6:30	7:00	8:00	12:00	16:00	1d:00:00	2d:00:00
17:17	17:25	17:30	17:45	18:00	19:00	20:00	1d:00:00	1d:08:00	1d:16:00	2d:00:00

each one contains the list of holes that finish before that point and after the previous. The holes are classified by the number of intervals of reference points they cover. Thus, a hole that starts and finishes between the same reference points is said to have a *span* of one. In general, a hole that starts at or after reference point rp_i and finishes at or before reference point rp_{i+k} will have a span of k. However, holes with the same span may not have the same availability, as this depends on the computational power of each execution node, so they are further classified by availability levels.

The aggregation process is only possible if reference points, spans and availability levels have similar values in all the availability summaries that are to be aggregated together. The availability of every hole is approximated to the immediately lower power of two, and the span of a hole is limited to the number of reference points before the corresponding one. For the reference points, a set of values is generated so that summaries created at near moments have most reference points in common. Table I shows three examples of the generated reference points. Values have a near-exponential growth, with some variations that make them more "human-readable".

Periodically, every execution node provides its routing node parent with updated availability information, which aggregates it with the information of the rest of its children to create the summary of its branch. Then, the new summary is sent to the next level of the tree and the process is repeated until the root is reached. Without a limit in this propagation, the root would be quickly flooded with update messages. This limit is implemented as a bound in the update messages traffic. Each update message has to wait at least for a configurable period of time after the last update message was sent. If another summary is generated while a message is waiting to be sent, the last one will be dropped in favor of the newer information.

C. Discovery and Allocation

The global scheduling algorithm decides how and where to route requests with task sequences, or whether they should be divided into smaller requests when sufficiently large holes are not found. When a routing node receives a request, it tries to look for a hole with enough availability to execute the whole sequence. It looks for holes that better fit the sequence requirements, leaving bigger holes in case a larger sequence is received later, and thus trying to improve resource usage. If no hole is found, the sequence is divided into two or more pieces, and the search is repeated on them.

In the end, if a hole is found for each piece, a new request is constructed for each one and sent to the corresponding subbranch. New startlines and deadlines are taken from the reference points that have been matched with task boundaries. If no hole has been discovered, the original request is routed to the next level of the tree to try further. Finally, if a request reaches the root node and no hole is found for it, it is returned to its owner, so that it can decide what to do with that sequence or with the whole workflow.

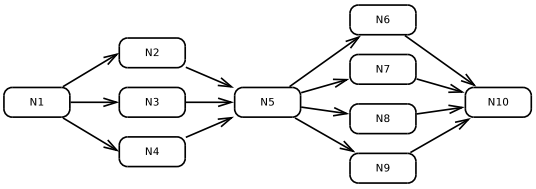

Fig. 4. A Fork-Join graph model.

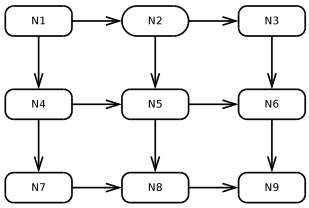

Fig. 5. A Laplace equation solver graph model.

VI. EXPERIMENTAL RESULTS

A. Simulation Environment

The simulated network overlay contains one million nodes with 1Mbps links, with a link delay of 50ms in mean, to model an average home Internet connection. The arrival of new workflows follows a Poisson process, with a mean interarrival time that maintains every node busy at almost any time. Finally, the deadline of each workflow is calculated through the *workflow relative priority* W_j, which is the ratio between the time the submission node would need to execute the workflow by itself and the time remaining until deadline. A value of 1 or less would mean that the sender could execute that workflow by itself, so higher values are tested. As workflows, two different kinds of DAGs has been selected to get results on different sizes and workflow widths. The selected models are a Fork-Join and a Laplace equation solver-like graph, shown in Figs. 4 and 5.

B. Test Results

Probably, the system property that users firstly notice is response time. In PeerComp, this property depends on two

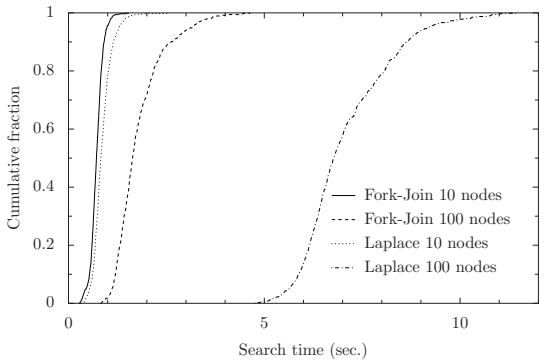

Fig. 6. Allocation time for different workflow widths, in a network of 1 million nodes.

TABLE II
AVERAGE SPEEDUP BY WORKFLOW MODELS AND PRIORITIES.

		Workflow relative priority W_j			
Model	Max. Speed-up	1.1	1.2	1.3	1.4
Laplace	1.8	2.05	2.09	2.18	2.22
Fork-Join	3.3	2.12	2.85	2.90	2.92

factors: workflow allocation time and workflow execution time. By allocation time we mean the time lapse between the moment the request is sent and all the tasks are allocated to an execution node. The main factor that decides the time a workflow needs to be allocated is its width. Fig. 6 shows the cumulative distribution of the allocation time for both models and different sizes. As expected, the allocation time of fork-join workflows varies very little with their size, while it grows significantly for Laplace-like models, which concurrency is lower. In any case, allocation is done in just a few seconds.

The workflow execution time is the time lapse between a request is sent and the last task is finished. We define the *speed-up* as the ratio between the time the submission node would need to execute the workflow by itself and the actual execution time. The ratio between workflow total length and minimum length provides the maximum speed-up that can be expected in execution nodes with similar computing power as the submission node. Table II presents average speed-up values registered for different workflow relative priorities. As it can be seen, the fork-join model has a maximum expected speed-up of 3.3 while the Laplace model could get a speed-up of 1.8. Results show that for the Laplace model, the system performs better than expected, as tasks may run in machines which are faster than the client's one. However, in the fork-join model the overhead of PeerComp makes it unable to reach the maximum expected speed-up. In any case, the speed-up increases with the workflow priority as deadlines become tighter.

Finally, we present the computational cost and the network traffic of the protocols. For the former, a message of any protocol was processed in less than 1ms, and a mean of just 2 events per second were processed by each node. Furthermore, the maximum recorded value was just 40 events per second, thus the computational overhead is considered negligible. For the later, the network traffic generated by PeerComp is also very reduced, mainly because of the update rate limitation. Thus, we see that about 75% of the nodes only generate less than 2500 Bps of peak traffic, with a registered maximum of 8200 Bps. Likewise, about 75% of the nodes receive less than 1000 Bps of peak traffic, just 1% of the nodes receive more than 8000 Bps, and the registered maximum during the tests was 29000 Bps.

VII. CONCLUSION

In this paper, a distributed scheduler of workflows with deadlines in a P2P computing platform has been presented. Its completely decentralized model has been validated with simulations that have shown fast response times and low overhead in a system with one million nodes. Some types of workflows, like Fork-Join ones, show schedule times of just a few seconds with this system size. Big workflows with highly concurrent tasks can be easily scheduled with low overhead and a good speed-up.

In future work, other policies can be evaluated in both scheduling levels, global and local, so that other features could be provided by the scheduler, like fairness or priorities.

ACKNOWLEDGMENTS

The research work presented in this paper has been supported by the CICYT DPI2006-15390 project of the Spanish Government, grant B018/2007 of the Aragonese Government, and the GISED, group of excellence recognised by the Aragonese Government.

REFERENCES

[1] G. Juve and E. Deelman, "Resource Provisioning Options for Large-scale Scientific Workflows," in *Third International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES)*, 2008, pp. 608–613.

[2] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. G. D. Okaya, K. Beattie, and T. Jordan, "Reducing Time-to-solution Using Distributed High-throughput Mega-workflows - Experiences from SCEC Cybershake," in *Fourth IEEE International Conference on e-Science (e-Science 2008)*, 2008, pp. 151–158.

[3] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor: A Distributed Job Scheduler," *Beowulf cluster computing with Linux*, pp. 307–350, 2002.

[4] F. Dong and S. G. Akl, "Distributed Double-level Workflow Scheduling Algorithms for Grid Computing," *Journal of Information Technology and Applications*, vol. 1, no. 4, pp. 261–273, 2007.

[5] R. Ranjan, M. Rahman, and R. Buyya, "A Decentralized and Cooperative Workflow Scheduling Algorithm," in *8th IEEE International Symposium on Cluster Computing and the Grid*, 2008, pp. 1–8.

[6] J. Celaya and U. Arronategui, "Scalable Architecture for Allocation of Idle CPUs in a P2P Network," *Lecture Notes in Computer Science*, vol. 4208, pp. 240–249, 2006.

[7] J. Celaya and U. Arronategui, "YA: Fast and Scalable Discovery of Idle CPUs in a P2P Network," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, 2006, pp. 49–55.

[8] H. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, "VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes," in *Proceedings of the 22nd International Conference on Data Engineering, 2006. ICDE '06*, 2006, p. 34.

[9] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task Scheduling Strategies for Workflow-based Applications in Grids," in *IEEE International Symposium on Cluster Computing and the Grid*, 2005, pp. 759–767.