

A Middleware for Job Distribution in Peer-to-Peer Networks

Thomas Fischer, Stephan Fudeus, and Peter Merz

University of Kaiserslautern, Kaiserslautern, Germany
fischer@informatik.uni-kl.de

Abstract. Recently, peer-to-peer technology has become important in designing (desktop) grids for large-scale distributed computing over the Internet. We present a middleware for distributed computing based on peer-to-peer systems. Our peer-to-peer desktop grid (P2P Grid) framework includes an efficient and fault-tolerant communication scheme for job distribution combining epidemic algorithms with chord-style multicasts. We show that this hybrid scheme is more efficient than both epidemic algorithms and chord-style multicasting alone. Moreover, we demonstrate that our middleware is capable of handling two different types of jobs: jobs comprised of independent tasks as well as jobs comprised of cooperating (communicating) tasks.

1 Introduction

Desktop grids allow the utilization of available computing power of modern desktop computers. Systems such as SETI@home [1], Folding@Home [2], Distributed.net [3] are examples of desktop grids in which a central server coordinates the distribution of tasks among available workers. The main drawback of this approach is that the central server may become a bottleneck and it represents a single point of failure. This problem has been addressed in [4], where a tree of scheduling nodes splits the job into subtasks before assigning subtasks to processing nodes.

An alternative approach is used in *Peer-to-Peer* (P2P) networks [5] consisting of a set of independent nodes (*peers*), which may be heterogeneous regarding their capabilities. Strict P2P systems have no dedicated nodes, less stricter P2P approaches use the concept of super-peers [6]. The advantages of P2P networks are their scalability, robustness and ability to self-organize; no (central) administration is needed.

P2P systems are mostly known for file-sharing applications such as Gnutella or Kazaa [7]. But the general advantages of P2P systems and the interest for desktop grids such as SETI@home [8] motivate the approach to combine both techniques in a single middleware as done e.g. in [9]. In this paper we present a middleware for scientific distributed computing following strictly the P2P paradigm. Our middleware allows any P2P node to initiate distributed computations that get efficiently distributed (both code and parameters) among participating nodes. Plus, jobs comprised of communicating tasks are supported as required for example in distributed evolutionary algorithms (DEA) [10].

2 Middleware Design

Our middleware allows the execution of scientific distributed algorithms including code and parameter distribution in a heterogeneous, dynamic and unreliable P2P environment. Existing distributed algorithms can be easily integrated by implementing simple Java interface classes. Compared to [9], we use a “flat” organization without hierarchy or node grouping. Furthermore, with our middleware inter-node communication is possible allowing it to run cooperating algorithms (e.g. island model based distributed evolutionary algorithms).

As implemented in our middleware, neighbor lists are exchanged using an *epidemic algorithm* [11], which provides a method for distributing information over a given network using the *gossip dissemination* paradigm. This way, the P2P overlay network is maintained: Nodes exchange these lists regularly, received lists will be merged locally (integrating new nodes) and outdated entries will be removed. Epidemic algorithms are resistant against obliteration, but permanently cause network traffic.

For processing, a job will be split into a set of subtasks, each describing one node’s task in the computation, all being either different (e.g. disjunct data sets) or identical (e.g. using stochastic processes in the distributed algorithm). To distribute a job (set of subtasks), we propose two algorithms plus a hybrid form, all resembling the following general distribution scheme: The initiator node inserts a set of subtasks into its subtask queue. Nodes regularly check their queue and, if not empty, either take one subtask for local processing or forward half of the subtasks to a neighbor node. Node failures are detected by missing subtask results at the initiator, which reissues the affected subtasks.

Within the *epidemic job distribution* scheme, a randomly selected neighbor node is the forwarding target. To ensure that all subtasks will be processed eventually, each node will take one element after another and process this subtask in parallel with the job distribution. For a better load-balancing, nodes may reject incoming subtasks by not sending an acknowledgment. Thus, subtask duplication is possible, when the acknowledgment is received too late at the sending node, which will resent the subtasks assuming subtask rejection at the receiver.

Reducing the epidemic job distribution’s overhead, an *efficient job distribution* scheme [12] is built on top of the epidemic algorithm. Every node selects an unique id allowing node arrangement in a logical ring. When initiating a new job, the first half of the queued subtasks will be sent to a node at the ring’s opposite side. Subsequently, each node sends half of its queue to the middle node of the remaining ring segment. This structured approach is faster (only $\log(n)$ time steps and $n - 1$ messages) than the unstructured epidemic algorithm, but it may not be complete, as a node may not know any appropriate nodes to send subtasks to, forcing the node to process all remaining subtasks locally.

This problem is addressed by the proposed *hybrid job distribution* scheme. Whenever the efficient chord-style distribution fails to send its remaining subtasks to other nodes on the ring, the subtasks will be handed over to the epidemic job distribution scheme avoiding that a single node has to process a larger number of subtasks if there are any free nodes known to the overloaded node.

3 Experiments

Our experiments have been performed on a Linux cluster consisting of 16 processors (3.0 GHz Pentium 4). On each computer, 10 middleware instances were started with different process priority levels to simulate an overlay network consisting of 160 heterogeneous nodes. One of the 160 instances was predetermined to be the first contact node for all nodes, another node was initiating a new job.

For the first set of experiments, three parameters have been varied: (1) The number of subtasks was either 64 or 256. (2) The size limit for the neighbor host list of each middleware instance was set to either 10 or 160. (3) The used job distribution scheme was either the epidemic, chord-style or hybrid approach.

In the second set of experiments, we ran (a) a fractal computation as an example for a job based on independent tasks as well as (b) a distributed evolutionary algorithm [10] for the traveling salesman problem (we considered a TSP instance containing 9847 Japanese cities).

3.1 Results

In Fig. 1, the results of the first set of experiments is displayed. The efficient job distribution requires little time to spread a job, but does so less equally leaving many subtasks at few nodes resulting in higher computation times. The epidemic approach requires more time for the job distribution, but as more nodes participate in the computation, the total time is lower compared to the first approach. Finally, the hybrid approach further decreasing the maximum number of subtasks per node. In Fig. 2, the results of the second set of experiments is displayed. For the fractal computation holds that the more subtasks a job is split into, the less total time is required, which is due to a better load-balancing when distributing many small subtasks among the nodes. For the distributed evolutionary algorithm solving TSP instances positive effects of the cooperation can be observed, as the tour quality increases with the number of nodes when keeping the number of iterations constant.

No. Subt.	Neigh. List	Distrib.	Time [s] to		No. Nodes		No. Subt.
			distrib.	finish	involved	comp.	max.
64	10	Efficient	.3	300.6	78.3	59.6	3.0
		Epidemic	7.5	107.6	63.0	64.0	2.0
		Hybrid	4.6	126.7	78.5	63.8	1.3
160	10	Efficient	.2	260.5	78.0	58.5	2.6
		Epidemic	7.4	107.5	63.2	64.0	2.0
		Hybrid	6.8	106.9	78.7	64.0	1.0
256	10	Efficient	1.0	1902.2	116.6	111.8	19.5
		Epidemic	114.4	214.4	149.1	150.0	3.0
		Hybrid	112.7	212.8	149.8	150.0	2.3
160	160	Efficient	13.9	1522.0	120.2	115.6	16.0
		Epidemic	113.3	213.3	149.5	150.0	3.0
		Hybrid	110.2	220.2	149.6	150.0	2.7

Fig. 1. Experimental results for different job distribution setups. The first three columns describe setup parameters. Results are averaged over ten runs.

No. Subt.	Fractal		TSP	
	Time [s] to finish	Best tour len after 100 iterations	Best tour len	after 100 iterations
16	51.7	516261 (4.95%)		
32	43.1	515793 (4.85%)		
64	30.7	515036 (4.70%)		
128	22.2	514964 (4.68%)		

Fig. 2. Results for the distributed fractal computation (time until all subtasks have returned their results, 10 run avg.) and the DEA for the TSP (best tour length and distance to optimum, 6 run avg.).

4 Conclusion

A middleware has been presented for distributed computing in desktop grids based on peer-to-peer overlay networks. The middleware cooperates a hybrid information/job distribution scheme combining fault-tolerant epidemic algorithms and efficient chord-style information propagation. In experiments, we have shown the effectiveness of this hybrid scheme compared to epidemic or chord-style job distribution alone. Moreover, we have demonstrated that the middleware is capable of executing fractal computations and hence jobs consisting of independent tasks as well as distributed evolutionary algorithms and hence jobs comprised of independent cooperating tasks very efficiently.

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM* **45**(11) (2002) 56–61
2. Larson, S.M., Snow, C.D., Shirts, M., Pande, V.S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics* (2002)
3. Distributed.Net: The Fastest Computer on Earth. <http://www.distributed.net/> (2006)
4. Page, A., Keane, T., Allen, R., Naughton, T.J., Waldron, J.: Multi-Tiered Distributed Computing Platform. In: PPPJ '03: Proceedings of the 2nd international conference on Principles and practice of programming in Java, New York, NY, USA, Computer Science Press, Inc. (2003) 191–194
5. Clark, D.: Face-to-face with peer-to-peer networking. *Computer* **34**(1) (2001) 18–21
6. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering (ICDE). (2003)
7. Leibowitz, N., Ripeanu, M., Wierzbicki, A.: Deconstructing the kazaa network. In: WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications, Washington, DC, USA, IEEE Computer Society (2003) 112
8. Anderson, D.P., Korpela, E., Walton, R.: High-Performance Task Distribution for Volunteer Computing. In: First IEEE International Conference on e-Science and Grid Technologies, Melbourne, Australia (2005)
9. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I.: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In Parashar, M., ed.: GRID '02: Proceedings of the Third International Workshop on Grid Computing. Volume 2536 of Lecture Notes in Computer Science., London, UK, Springer-Verlag (2002) 1–12
10. Bäck, T.: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York (1996)
11. Portmann, M., Seneviratne, A.: Cost-effective broadcast for fully decentralized peer-to-peer networks. *Computer Communications* **26**(11) (2003) 1159–1167
12. Merz, P., Gorunova, K.: Efficient Broadcast in P2P Grids. In: Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK (2005)