

A Scalable and Robust Framework for Distributed Applications

Márk Jelasity

Department of Artificial Intelligence, Free University of Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, jelasity@cs.vu.nl
and RGAI, University of Szeged, Hungary

Mike Preuß

Chair of Systems Analysis, Department of Computer Science, University of Dortmund
Joseph-von-Fraunhoferstr. 20, 44227 Dortmund, Germany, mike.preuss@uni-dortmund.de

Ben Paechter

Napier University, 10 Colinton Road, Edinburgh, Scotland, EH10 5DT

Abstract—This paper describes a novel tool for running distributed experiments on the Internet. The possible applications include simple load balancing, parallel evolutionary computation, agent-based simulation and artificial life. Our environment is based on cutting-edge peer-to-peer (P2P) technology. We demonstrate the potentials of the framework by analyzing a simple distributed multistart hillclimber application. We present theoretical and empirical evidence that our approach is scalable, effective and robust.

I. INTRODUCTION

This paper describes a novel tool for running distributed experiments on the Internet. It is being developed as part of the DREAM project (see project statement [1]). In a nutshell, the aim of the DREAM project is to develop a complete environment for developing and running distributed evolutionary computation experiments on the Internet in a robust and scalable fashion. Accordingly, the project involves developing different libraries, user interfaces and tools to support application development and execution. The present work focuses on the network engine, i.e. the overlay network on which these experiments will eventually be run.

Although our project focuses on evolutionary computation the environment supports any application which

- is massively parallelizable
- is asynchronous
- has little communication between its subprocesses
- has large resource requirements
- and is robust (the success of the application does not depend on the success of any given subprocess).

This list might seem quite restrictive but in fact it includes many interesting fields. Good examples are running independent tasks with load balancing, island models in evolutionary computation (EC), heuristic optimization, modeling swarm intelligence and other systems with emergent behaviour, etc.

The above assumptions essentially involve the relaxation of the strict reliability requirement and the burden of synchronizing and controlling subprocesses. This allows us to apply P2P

This work is funded as part of the European Commission Information Society Technologies Programme (Future and Emerging Technologies). The authors have sole responsibility for this work, it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein.

technology, i.e. technology which does not assume any central and guaranteed control mechanism or database. Instead it is based only on local information and local communication between the components of the network.¹

P2P approaches provide only statistical guarantees for reliability but work on wide area networks (WANs) where the costs of communication are much higher than within a parallel machine (cluster) or local network. The advantage of targeting WANs is the *huge amount of resources* that is available in the form of e.g. the idle CPU time of computers with an Internet connection. Another advantage of applying P2P technology is *scalability*: the environment can easily grow with the rapid growth of the amount of resources connected to the Internet without any further investment.

To the authors' knowledge, such an application of P2P technology for distributed computation is new. Existing P2P systems are used only to maintain discussion groups or to distribute information or data (e.g. [4], [5], [6]). Systems that use WANs for solving computational problems are not P2P and therefore are not scalable (i.e. growing needs much effort and investment) (e.g. [7], [8], [3]). The java platform offers a natural possibility to distribute computational tasks through allowing runtime linking of executable code to an application. It provides rich security features and at last but not least complete *platform independence*. This made Java an obvious choice for us.

To summarize: our environment can be thought of as a virtual machine or distributed resource machine (DRM) made up of computers anywhere on the Internet. The actual set of machines can constantly change and can also grow to a (very huge) theoretical limit discussed in Section II without any special intervention. Apart from security considerations, anyone having access to the Internet can connect to the DRM and can either run his/her own experiments or simply donate the spare capacity of his or her machine.

The outline of the paper is as follows: Section II discusses the DRM from an algorithmic and theoretical point of view. We will illustrate the scalability and robustness of the underlying

¹Note that there are broader interpretations of P2P technology, with emphasis on only resource-sharing or on the existence of some local communication between participants. For example Napster [2] and even United Devices [3] are P2P systems in this broader sense but not in the sense of our definition.

name	this is the unique key
address	the IP address and port of the node
date	timestamp of the entry
agents []	names of agents living at the node
map	optional information in a hash map

TABLE I
STRUCTURE OF AN ENTRY IN THE DATABASE OF A NODE.

epidemic protocol.

Section III describes an application developed for this framework. In this application the multistart stochastic hillclimbing (MSHC) algorithm is parallelized; hillclimbers from random starting points are distributed over our network using an algorithm developed by us. While this is only a simple application and does not make use of all the possibilities it is suitable for illustrating the features of the DRM. We should note that in spite of its simplicity the MSHC algorithm is sometimes surprisingly successful so this example has considerable practical relevance as well [9], [10]. The section contains the algorithmic description and also theoretical analysis of the effectiveness of the applied distribution scheme.

Section IV describes the results of our experiments on a real DRM under different circumstances: in a stable, in an unstable, and in a suddenly changing environment. Finally Section V concludes the paper.

II. THE DISTRIBUTED RESOURCE MACHINE

The DRM is a P2P overlay network on the Internet forming an *autonomous agent environment*. The applications are implemented as multi-agent applications. The exact way an application is implemented in the multi-agent framework is not restricted in any way (only by the available features and functionality of course), although we intend to suggest templates and examples in the future (one of which is discussed in Section III) to facilitate development.

For example in the EC domain a natural choice is to implement island models via a set of agents, each running an island of evolution, and the information exchange (migration) can be implemented on top of the communication functionality (messaging) of the DRM. The extension libraries being developed by our project support such higher level programming templates.

A. Self-Organizing Structure

The DRM is a network of DRM nodes. In the DRM every node is completely equivalent. There are no nodes that possess special information or have special functions. In other words in the DRM there are no servers. Yet the nodes must be able to know enough about the rest of the network in order to be able to remain connected to it and to provide information about it to the agents. The mechanism that allows this is a so called *epidemic protocol* [11].

Every node maintains an incomplete database about the rest of the network. This database contains entries on some other nodes (see Table I). We call these nodes *the neighbours* of the node. The database is refreshed using a push-pull anti-entropy epidemic algorithm. This algorithm works as follows: every

node s chooses a living address from its database regularly once within a time interval. An address is *living* if there is a working node s' at that address (not necessarily the same node as in the entry with the living address: it might be a new node started on the same address). Then any differences between s and s' are resolved so that after the communication s and s' will both have the union of the two original databases (choosing the fresher item if both contained items with a given key). Besides this s will receive a fresh item on s' (with a new timestamp of course) and s' will also receive an item on s with the actual timestamp.

The idea behind such algorithms is that in this way new information “infects” the network by getting in contact with an (initially) exponentially growing front of the nodes (see also Section II-B).

Two additions have to be made to this algorithm. The first is that entries older than a given age have to be removed from the database. The second is that the size of the database has to be limited, otherwise, in the case of huge networks, the database also becomes huge making the anti-entropy algorithm impractical. Fortunately the theoretical and practical results discussed below show that both can be done without sacrificing the power of the epidemic algorithm.

To connect a new node to the DRM one needs only one living address. The database of the new node is initialized with the entry containing the living address only, and the rest is taken care of by the epidemic algorithm described above. Removal of a node does not need any administration at all. Note that a node might even change its IP address and/or port while running, so computers with dynamic IP addresses are also automatically supported without any special modification of the algorithm.

B. Theoretical Properties

Since the epidemic algorithm is essentially statistical in nature it is important to understand its behaviour well. The following paragraphs will give answers to naturally arising questions.

An important assumption of the following results will be that a given node has an equal probability of being in the database of any other node. We will not analyse this assumption in this paper. Let us mention however that the protocol is completely location blind which would prove this assumption if the network connections were reliable. Unfortunately they are not thus the exact effect of the degree of reliability (and possibly the topology of the underlying networks) should be studied in the future.

In all the following discussions the following notation will be used: n is the number of nodes in the network, k is the size of the database in each node and a node initiates exactly one information exchange session in every t seconds.

B.1 How long does it take for a new entry to reach a given node?

Let us assume that new information emerges in the system, e.g. a new agent arrives at one of the nodes. Let the time be 0 when introducing the new information in the system. Let p_i be the probability that at time point it the fixed node s has not yet received the information. If we consider the pull version of the epidemic algorithm then $p_{i+1} = p_i^2$ clearly holds since s did not know the information at time it and in step $i + 1$ contacted another node which has not yet learned about it either [11]. It is

elementary to show that for any given constant c

$$p_{i(c,n)} < e^c, \quad i(c,n) = O(\log n)$$

Finally note that the push-pull version used in our system is of course at least as fast as the plain pull variant.

B.2 How large the database must be to ensure connectivity?

We know already that information spreads very fast over the network *if* the network is connected. But what is the probability that the network is connected? The answer is given in [12] using results and techniques from random graph theory.

Let $\mathcal{G}(n, k)$ denote a random directed graph of n nodes in which the outdegree of each node is exactly k and these k arcs go to random nodes. Let $\pi(k, n)$ denote the probability that there is a directed path from a given node to any other node in $\mathcal{G}(n, k)$. The following theorem holds:

Theorem 1: Consider the sequence of random graphs $\mathcal{G}(n, k_n)$ with $k_n = \log n + c + o(1)$, where c is a constant. We have

$$\lim_{n \rightarrow \infty} \pi(k_n, n) = e^{-e^{-c}}$$

It is notable that $1 - \pi(k_n, n) < 10^{-10}$ if $n > 23$. The theorem tells us that for a large network of size n if the size of the database is $k = \log n + c$ where e.g. $c > 23$ we have a connected network with an extremely large probability. For example for $k = 100$ we can have $n \approx 10^{33}$. Empirical analysis proves that the constants predicted by the theorem provide the expected performance [12], [13].

B.3 Is it possible to have a “traffic jam”?

A given node s initiates exactly one information exchange session in every t seconds. However one might suspect that since other nodes can initiate sessions to s as well, in case of large networks we can have a congestion if too many other nodes want to talk to s at the same time. Fortunately this has a very low probability. Let x be a random variable denoting the incoming requests during a fixed time interval of t seconds. Assuming random distribution of the connections in our network it is easy to see that the distribution of x is binomial with the parameters $n - 1$ and $p = 1/(n - 1)$. Accordingly, the expected value and variance are

$$\mu = \frac{n-1}{n-1} = 1, \quad \sigma^2 = (1-p)\mu = \frac{n-2}{n-1} < 1$$

From around $n > 50$ this distribution can be very closely approximated by the Poisson distribution with parameter $\lambda = 1$ (independently of n). For this distribution we have e.g. $\Pr(x > 6) < 10^{-4}$ and $\Pr(x > 12) < 10^{-10}$. Given the relatively small required size of the databases at each node (as proved above) this can be easily handled. Choosing a large value for t (say one minute) can reduce communication even further. Recall that this bounds are independent of n so scalability is not damaged.

III. THE TEST APPLICATION

The application itself has two layers. One layer is an abstract load balancing framework on top of the DRM which has very interesting characteristics in spite of its simplicity. The second (highest) layer is the subset sum problem—the actual problem to be solved—discussed in Section III-B.

A. The Load Balancing Algorithm

In this paper we chose to consider the simplest possible application on the DRM, a load balancing framework. This framework does not make use of the messaging features of the DRM (at least not on the application level) i.e. the tasks do not communicate with each other. The reason for this choice is that this work focuses on the DRM and this application suffices for illustrating the reliability, scalability and robustness of the system.

We assume that our application is composed of T tasks that have to be run independently of each other as fast as possible. The tasks have to be distributed efficiently over the available resources in a way that tolerates the unreliability and high communication costs of WANs, and the dynamic nature of the DRM, i.e. that machines can come and go at any time.

Load balancing is based on epidemic algorithms just like the DRM itself. The application starts by starting a so called *island* which is in fact an autonomous agent. This island can be started on any node within the DRM. The goal of this island is to complete T tasks. The island achieves this goal by starting to work on a task and at the same time listening to the communications of its host node. When the node exchanges information with another (according to the epidemic protocol of the DRM) the island checks if the peer node has an island already (recall that the database item of a node contains information about the agents running there). If not it sends half of its remaining tasks to the peer node in the form of a new island which then runs the same distribution mechanism on the peer node in order to complete its tasks. In this way the tasks “infect” the network. Note that it means that there is at most one island on every node.

When an island arrives at its host node it sends a confirmation message back. This is the only communication that is taking place. It ensures that the processing of the set of tasks sent to another node was at least started. Confirmation of finishing tasks does not make sense since the sender island might not exist anymore at that time. This might result in losing tasks but this is not a serious disadvantage under our assumptions described in Section I.

Let us briefly mention two notable properties of this algorithm. Firstly, the performance of the nodes does not have to be taken into account when sending out tasks since if the machine is too slow, it will send most of its tasks on anyway to nodes that finish earlier. Secondly, no mechanism is needed to indicate that free resources are available because those resources will be infected very quickly anyway by simply communicating with peers. These properties are possible due to the epidemic protocol underlying the DRM discussed earlier.

A.1 Some Theoretical Properties

Let T be the number of tasks and n the number of nodes. We have to differentiate between three cases.

A.1.a Huge network: $T \ll n$. Here the probability p^* that an island with more than one task can send some of its tasks on when contacting (or being contacted by) another node is $p^* \geq p = (n - T)/n$ which is close to 1. Assuming $p^* = p$ the distribution of the number of communications needed to find a free node follows a geometric distribution with the parameter p

with the expected value and variance

$$\mu = \frac{1}{p} = \frac{n}{n - T}, \quad \sigma^2 = \frac{1 - p}{p^2}$$

These are close to 1 and 0 respectively. This—together with the fact that the islands send half of their tasks every time they can—means that the tasks literally explode into the space of the DRM resources. The time needed to completely distribute the tasks is $O(\log T)$. Recall that we had huge networks in mind when designing the DRM.

A.1.b Small network: $T \gg n$. In this case the expected number of tasks an island has is much more than one. When in such a network an empty node connects to a random peer (according to the epidemic algorithm) it is very likely that this peer will have some tasks to send. Thus it is very unlikely that any node remains empty for a long time. This situation is analogous to the end-phase of the pull version of the epidemic algorithm when most of the nodes already know some piece of new information. A more exact formulation can be found in [11].

A.1.c $T \approx n$. This case (unlike the first two) is not optimal. The reason is that a point comes when a considerable number of islands work only on one task. At this point these islands occupy a considerable proportion of the DRM. Because of this the islands with more tasks find the islands which have none slower. However in the first phase the growth is fast, it slows down only at the end phase. But even in this case if the completion time of an average task is much longer than the time between two information exchanges between the nodes then this disadvantage becomes almost invisible.

B. The Subset Sum Problem

Even though the actual problem we solve does not play a major part we chose to use a real problem instead of running sleeping cycles or benchmarking code. We would like to emphasize however that this paper is not about solving the subset sum problem. For that purpose metaheuristics like evolutionary algorithms or different hillclimbers are not the best choice anyway as specific algorithms based on interesting mathematical results are known [14], [15]. For the sake of completeness we give the description of a task.

In the case of the subset sum problem we are given a set $W = \{w_1, w_2, \dots, w_n\}$ of n positive integers and a positive integer M . We would like to find a $V \subseteq W$ such that the sum of the elements in V is closest to, without exceeding, M . This problem is NP-complete. This does not mean that every instance is hard. When the elements of W are relatively large the instances become harder. In particular, when the elements of W are uniformly drawn from $(0, 2^n)$ and M is close to the average of the set elements, the instance is very hard [14]. We used such an instance with $n = 100$.

A task was running a stochastic multistart hillclimber (SMHC) for a given number of evaluations. The result of a task is the best solution found. A solution (which is always a subset) was encoded as a binary vector of length n . The neighborhood for the hillclimber was defined by a usual mutation operator: every bit is flipped with a probability of $2/n$. In every step a new solution is generated. The new solution is accepted if it is

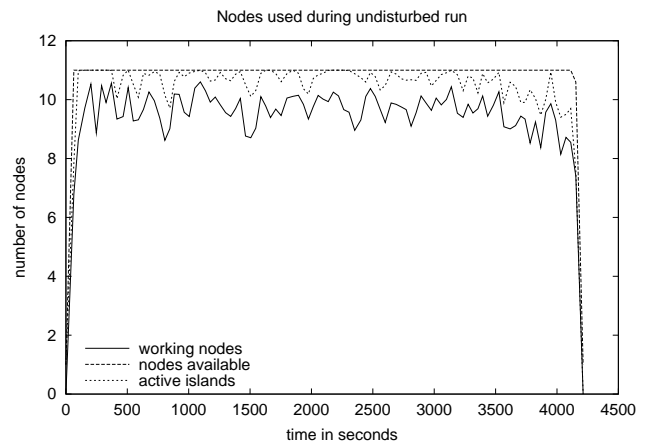


Fig. 1.

not worse than the actual solution (for evaluation the approach used in [16] was adopted). The search is restarted from a random solution if there is no improvement within a given number of evaluations.

IV. EMPIRICAL RESULTS

We designed three different scenarios to show that what has been claimed concerning DRM and epidemic load balancing properties holds when run on a cluster of real machines. For each of them, a fixed number of tasks (here: 1500) was computed.

Optimistic scenario: The experiment is running on an undisturbed cluster, no node is added, deleted or restarted.

Cluster addition scenario: After the experiment has been running for a time, a huge number of nodes is added to and deleted from the DRM several times. The added nodes are always empty, i.e. they have no tasks.

Intensified real-world scenario: A number of nodes is intentionally made unstable. Each of the modified nodes resets and starts all over again several times.

We are interested in the performance behaviour of the DRM under these different conditions, and would like to see a speedup factor that does not vary much over the three scenarios. Note that we do not want to show robustness in the sense of getting all of the tasks done, this is hindered by the layout of the load balancing system and would in any case be very difficult to achieve on a large-scale distributed p2p system. From our experiments, we cannot conclude much about scalability of the performance behaviour because the number of machines available for testing has been quite limited.

It must be stated that even for the optimistic scenario it is very difficult, if not impossible, to repeat an experiment in exactly the same way. However, we consider the average results over many experiments relatively stable.

A. Optimistic Scenario

The number of nodes is stable for this scenario, the experiment runs undisturbed by external influences. This can easily be confirmed visually from Fig. 1. It also shows that the number of working nodes varies between 9 and 11 until most of the

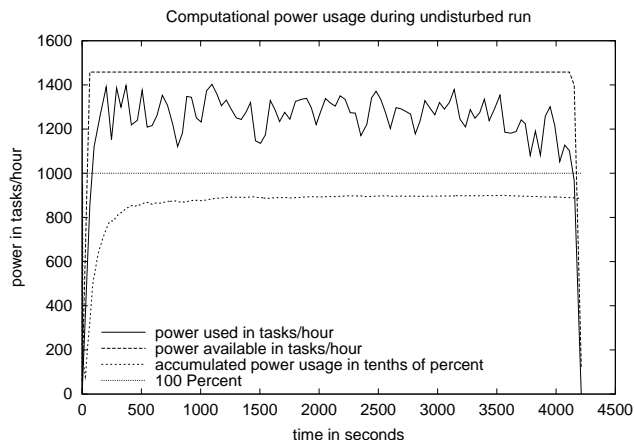


Fig. 2.

tasks are done. At around 3500 seconds, the task distribution seems to get more difficult, so that rebalancing the system begins to take longer and more nodes remain without work. At this point, 83% of the tasks are done. But as long as the number of tasks available exceeds the number of nodes by far, the DRM can recover from this situation. Near the end, the number of tasks left to compute approaches the number of nodes and the suboptimal behaviour described in Section III-A.1 ($T \approx n$) appears as predicted. The slowest of the nodes that still have one task to compute determines the end of the experiment. In this case, all of the 1500 tasks have been executed. Note that the number of active islands differs slightly from the working nodes. That is because islands performing task setup and distribution are considered active, but their node is not considered working during this administration time.

Figure 2 shows a very similar structure. It displays the used resources relative to the available capacity in terms of tasks per hour. These numbers are determined by using the tasks themselves as a benchmark and computing the approximate maximum speed of a node via the average time needed to finish a task. The accumulated power usage shown as separate line proves that the available total capacity of all nodes in the experiment is used to more than 86%.

B. Cluster Addition Scenario

It can be visually perceived from fig. 3 that 9 additional nodes have been added to the DRM after around 300 seconds. They are quickly found and exploited by placing new islands on them. After 750 seconds have elapsed, the nodes are removed again by shutting them down. This is repeated with 10 nodes later on, this time removing them step by step and not at once. Despite the expectation that this scenario depicts a very extreme course, the DRM copes quite well with the situation. Available resources are utilised rapidly and even the deletion of half of the nodes does not hinder the experiment from continuing.

For this experiment, the difference between the two types of chart (Figures 3 and 4) is clearer. The reason is that the capacity of the added nodes is lower than the capacity of the starting nodes. This is indicated by the smaller steps visible in Fig. 4. Both charts suggest however that most of the available resources

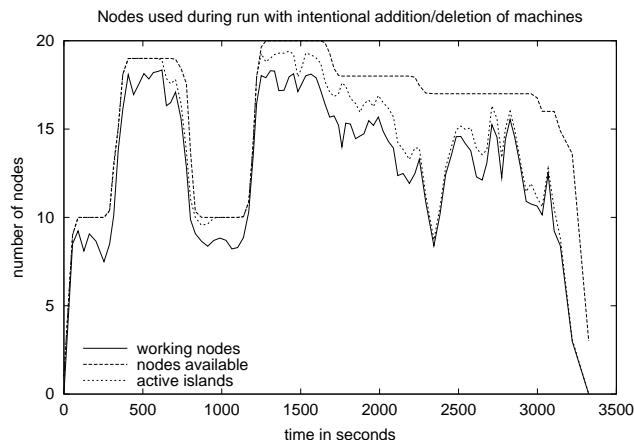


Fig. 3.

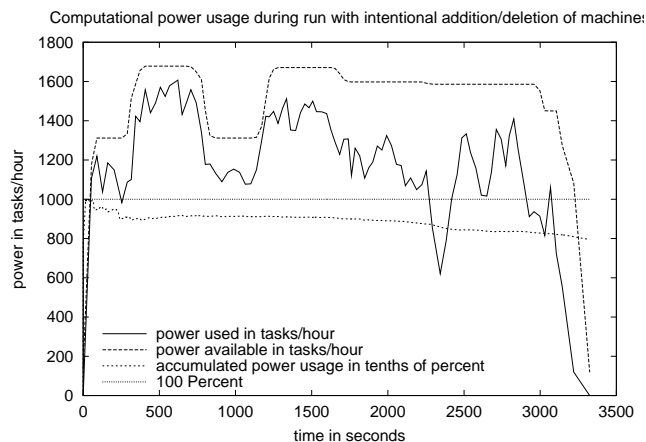


Fig. 4.

are used. At the end, the accumulated power usage is 80%. It is however important to note that not all tasks are actually completed in this scenario. As the islands own their tasks after confirmation (they are not memorized anywhere else in the DRM), the tasks of a prematurely shut down island are lost. Thus, the number of tasks completed in this experiment is only 1104 of 1500. This result would have been different if only the network connection of the additional nodes had been disabled. But after restart, they have no knowledge of any previous task or island.

C. Intensified Real-World Scenario

In this scenario, 10 of the 19 used nodes are made unreliable (Figures 5 and 6). This is modelled by enforcing a restart at a randomly chosen time Gaussian distributed around 600 seconds. Thus we assume that every one of the unreliable nodes is restarted several times during the experiment. As stated before, a node restart means loss of the actual tasks concluded in its island and therefore the number of finished tasks at the end of the experiment is expected to be quite low. In fact, only 661 of 1500 tasks are computed. But it should be noted that this scenario really is *extremely* intensified.

Nevertheless, the DRM again proves the ability to cope well

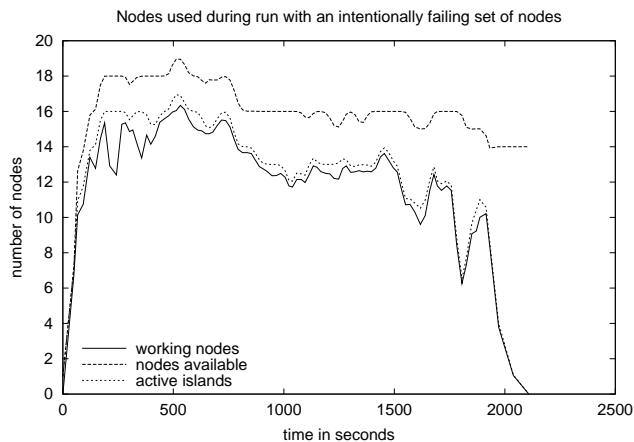


Fig. 5.

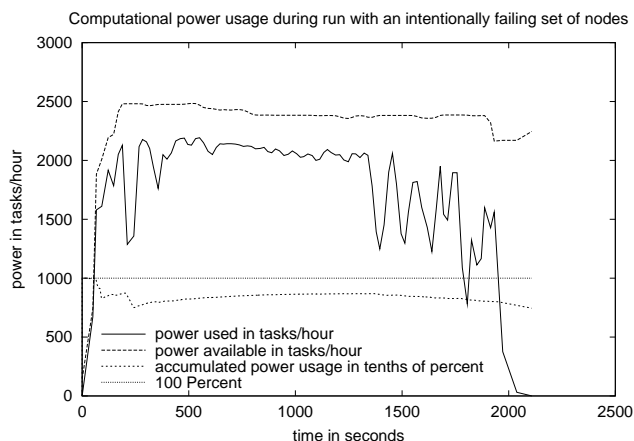


Fig. 6.

with the situation. One example may illustrate this behaviour. After 1800 seconds, a difficult task redistribution becomes necessary, indicated by the number of working nodes going down to 7. At this time, only 38 tasks are left in the system, thus the suboptimal behaviour described in Section III-A.1 ($T \approx n$) is near. But the load balancing algorithm places tasks on another 3 nodes. In the end, the accumulated power usage is 75%.

D. Results

For the sake of completeness we give the results of the optimization. For the meaning of the values below please refer to [16]. Over all the experiments performed the mean of the best objective function values found in the single hillclimbing runs vary from $1.41E25$ to $1.53E25$ and the best value we found is $8.51E20$.

V. CONCLUSIONS

In this paper we discussed a distributed P2P environment for running special distributed applications from domains like evolutionary computation, social modeling, artificial life, etc. We focused on the scalability and robustness of the environment.

Theoretical properties of the epidemic algorithm responsi-

ble for information flow and the connectivity of the system were discussed. These results show that under certain assumptions the required scalability and reliability are present. Large scale empirical simulations of the DRM are currently being conducted in order to gain more insight into the dynamics of the network.

Empirical results on a real network were also presented. Probably the simplest possible application (load balancing of a given number of independent MSHC optimizations) was chosen to illustrate the potentials of the system. The application reacted well to extreme conditions like fluctuation and sudden changes of the available resources.

ACKNOWLEDGMENTS

The authors would like to thank the other members of the DREAM project for fruitful discussions, the early pioneers [1] as well as the rest of the DREAM staff, Maribel García Arenas, Emin Aydin, Pierre Collet and Daniele Denaro.

REFERENCES

- [1] Ben Paechter, Thomas Bäck, Marc Schoenauer, Michele Sebag, A. E. Eiben, Juan J. Merelo, and Terry C. Fogarty, "A distributed resource evolutionary algorithm machine (DREAM)," in *Proceedings of the Congress on Evolutionary Computation 2000 (CEC2000)*. IEEE, 2000, pp. 951–958, IEEE Press.
- [2] Napster, "<http://napster.com/>," .
- [3] United Devicestm, "<http://ud.com/>," .
- [4] Peter Druschel and Antony Rowstron, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, 2001, ACM.
- [5] Gnutella, "<http://gnutella.wego.com/>," .
- [6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*, Hannes Federrath, Ed., 2000, vol. 2009 of *LNCS*, pp. 46–66.
- [7] distributed.net, "<http://distributed.net/>," .
- [8] SETI@home, "<http://setiathome.ssl.berkeley.edu/>," .
- [9] Andrew L. Tuson, Richard Wheeler, and Peter Ross, "Emergency resource redistribution in the developing world: Towards a practical evolutionary/meta-heuristic scheduling system," in *Proceedings of Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97)*, Glasgow, UK, 1997, IEE/IEEE.
- [10] Ari Juels and Martin Wattenberg, "Stochastic hillclimbing as a baseline method for evaluating genetic algorithms," in *Advances in Neural Information Processing Systems*, David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, Eds. 1996, vol. 8, pp. 430–436, The MIT Press.
- [11] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, "Epidemic algorithms for replicated database management," in *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, Vancouver, Aug. 1987, ACM, pp. 1–12.
- [12] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," Submitted for publication, available as <http://research.microsoft.com/camdis/PUBLIS/kermarrec.ps>.
- [13] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Anne-Marie Kermarrec, and Petr Kouznetsov, "Lightweight probabilistic broadcast," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001)*, Göteborg, Sweden, 2001.
- [14] Claus-Peter Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, pp. 181–191, 1994.
- [15] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern, "An improved low-density subset sum algorithm," *Computational Complexity*, vol. 2, pp. 111–128, 1992.
- [16] Sami Khuri, Thomas Bäck, and Jörg Heitkötter, "An evolutionary approach to combinatorial optimization problems," in *Proceedings of the 22nd annual ACM computer science conference on Scaling up: meeting the challenge of complexity in real-world computing applications (CSC'94)*. ACM, 1994, pp. 66–73.