DAxML: A Program for Distributed Computation of Phylogenetic Trees Based on Load Managed CORBA *

Alexandros P. Stamatakis[1], Markus Lindermeier[1], Michael Ott[1], Thomas Ludwig[2], and Harald Meier[1]

[1] Technical University of Munich, Department of Computer Science
Boltzmannstr. 3, 85748 Garching b. München, Germany
{stamatak, linderme, ottmi, meierh}@in.tum.de
wwwbode.cs.tum.edu

[2] Ruprecht-Karls University, Department of Computer Science
Im Neuenheimer Feld 348, 69120 Heidelberg, Germany
thomas.ludwig@informatik.uni-heidelberg.de
pvs.iwr.uni-heidelberg.de

Abstract. High performance computing in bioinformatics has led to important progress in the field of genome analysis. Due to the huge amount of data and the complexity of the underlying algorithms many problems can only be solved by using supercomputers. In this paper we present **DAxML**, a program for the distributed computation of evolutionary trees. In contrast to prior approaches **DAxML** runs on a cluster of workstations instead of an expensive supercomputer. For this purpose we transformed **PAxML**, a fast parallel phylogeny program incorporating novel algorithmic optimizations, into a distributed application. **DAxML** uses modern object-oriented middleware instead of message-passing communication in order to reduce the development and maintenance costs. Our goal is to provide **DAxML** to a broad range of users, in particular those who do not have supercomputers at their disposal. We ensure high performance and scalability by applying a high-level load management service called **LMC** (**L**oad **M**anaged **C**ORBA). **LMC** provides transparent system level load management by integrating the load management functionality directly into the ORB. In this paper we demonstrate the simplicity of integrating **LMC** into a real-world application and how it enhances the performance and scalability of **DAxML**.

1 Introduction

Within the framework of the **ParBaum** project at the TUM (Technische Universität München) work is conducted in the area of high performance bioinformatics

* This work is partially sponsored under the project ID **ParBaum**, within the framework of the "Competence Network for Technical, Scientific High Performance Computing in Bavaria": KONWIHR (Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern). KONWIHR is funded by means of "High-Tech-Offensive Bayern".

in order to design novel parallel and distributed systems as well as algorithms for large-scale phylogenetic (evolutionary) tree computations based on the maximum likelihood method.

Phylogenetic trees describe the relative evolutionary distances between organisms and are calculated using information from their genetic sequences. The rRNA (ribosomal RiboNucleic Acid) is a distinguished, highly conserved region of an organisms genetic sequence and is therefore apt for determining evolutionary relationships.

Our work relies on sequence data provided by the ARB [16] (Latin, "arbor" = tree) rRNA-sequence database, which provides a huge amount of high quality sequence data and is a joint development of the LRR (Lehrstuhl für Rechnertechnik und Rechnerorganisation) and the "Department of Microbiology" of the TUM. The ARB software is a graphically oriented package comprising various tools for sequence database handling and data analysis. A central database of processed (aligned) sequences and any type of additional data linked to the respective sequence entries is structured according to phylogenetic or other user defined criteria.

The maximum likelihood method renders evolutionary trees of high quality. A recent result by Korber *et al.* that times the evolution of the HIV-1 virus [3] demonstrates that maximum likelihood techniques can be effective and important for solving scientific problems in medicine and biology. However computing evolutionary trees based on this model is extremely computationally expensive. Thus, only relatively small trees ($\approx$ 500 sequences [14],[15]), compared to the huge amount of data available ($\approx$ 20000 sequences in today's databases), have been calculated on supercomputers so far.

Within this context we investigate different approaches for handling the complexity of the problem. In this paper we focus on the distributed computation of large phylogenetic trees.

An important property of existing parallel phylogeny programs, such as **parallel fastDNAml** [14] or **PAxML** [8], [10], [11], [12], [13], is that they are well suited for distributed computation, since the largest part of computation time is consumed by the workers during tree evaluation and comparatively small amounts of data are communicated in a simple string format. Furthermore, at each step of the computation there is a large amount of independent tasks that can easily be distributed among the workers.

For handling the complexity and heterogeneity of todays computing environments and to exploit the vast amount of unused resources one can typically find in organizations, such as universities or research laboratories the distributed object-oriented programming paradigm is the most adequate mechanism, especially coupled with a powerful load balancing tool. With **DAxML** (**D**istributed **A(x)**ccelerated **M**aximum **L**ikelihood) we present a new approach for the calculation of large phylogenetic trees exploiting the advantages of the distributed object-oriented programming paradigm through the integration of the powerful load management tool **LMC** paired with the very fast tree evaluation function of **PAxML** [10], [13], which is based on novel algorithmic optimizations.

2 The Load Management System

Nowadays applications do not reside on a single host anymore - they are distributed all over the world and interact through well defined protocols. Global interaction is accomplished by so called middleware architectures. The most common middleware architectures for distributed object-oriented applications are the CORBA (Common Object Request Broker Architecture) and the DCOM (Distributed Component Object Model). Environments like CORBA and DCOM cause new problems because of their distribution. A significant problem is load imbalance. As application objects are distributed over multiple hosts, the slowest host determines the overall performance of an application. Load management services intend to compensate load imbalance by distributing workload. This guarantees both, high performance, as well as scalability of distributed applications.

Our load management concept uses objects as load distribution entities and hosts as load distribution targets. Workload is distributed by initial placement, migration, and replication.

- *Initial Placement* stands for the creation of an object on a host that has sufficient computing resources in order to efficiently execute the object.
- *Migration* means moving an existing object to another host that promises a more expeditious execution.
- *Replication* is similar to migration but the original object is not removed, such that identical objects called replicas are created. Further requests to the object are split up among its replicas in order to distribute the workload (requests) among them.

There are two kinds of overload in distributed object-oriented systems - background overload and request overload. Background load is caused by applications that are not controlled by the load management system. Request overload means that an object is not capable to efficiently process all requests it receives. Migration is an adequate technique for handling background load but the scalability attained by migration is limited. Replication helps to break this limitations and is an adequate technique for handling request overload.

We implemented these concepts in the **LMC** system [5]. **LMC** is a load management system for CORBA. The main components of **LMC** are shown in Figure 1. These components fulfill different tasks and work at different abstraction levels. The load monitoring component offers both, information on available computing resources and their utilization, as well as information on application objects and their resource usage. This data has to be provided dynamically, i.e. at runtime, in order to obtain information about the runtime environment and the respective objects. Load distribution provides the functionality for distributing workload by initial placement, migration, or replication of objects. Finally, the load evaluation component decides about load distribution based on information provided by load monitoring. Those decisions can be attained by a variety of strategies, which are discussed in detail in [6].

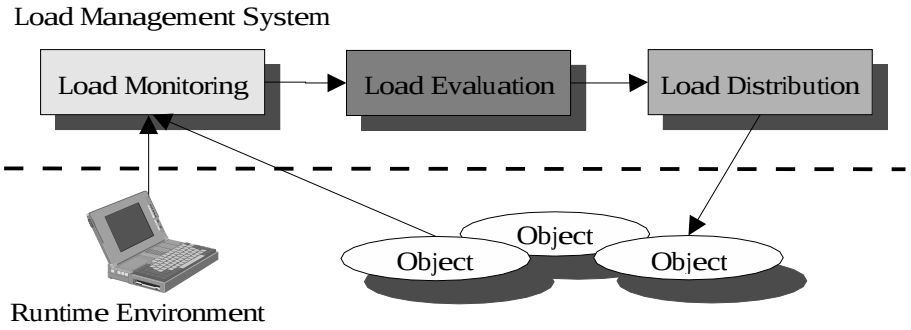

Fig. 1. The components of the Load Management System LMC

LMC is completely transparent on the client-side because it uses CORBA's Location Forward mechanism to distribute requests among replicas. On the server-side minor changes to the existing code are necessary for integrating load management functionality into the application. These changes mainly affect the configuration of the Portable Object Adapter (POA). All extensions are seamlessly integrated into the CORBA programming model. Thus, only a minor additional effort is required by the application programmer for the integration of the services provided by **LMC**.

For a detailed description of the load managment system as well as the initial placement, migration and replication policies used see [6].

3 The Application

DAxML is based on **PAxML**, which is in turn a derivative of the latest release of **parallel fastDNAml** (version 1.2.2). The essential difference between **parallel fastDNAml** and **PAxML** are several novel algorithmic optimizations of the topology evaluation function, which, depending on the input data set and the processor architecture, lead to global run time improvements between 27% and 65% [11], [13]. Note that **PAxML** scales particularly well on PC processor architectures, which are the main target platforms for **DAxML**. Since our optimizations are purely algorithmic **PAxML/DAxML** render exactly the same results as **parallel fastDNAml**.

The attained level of run time improvement attained by our algorithmic optimizations is significant (see Figure 3), because our work focuses on computations of huge phylogenetic trees.

Since the algorithmic optimizations introduced by **DAxML** are on a fine granularity level and do not affect the parallelization concept, we will restrain our analysis to a brief description of the sequential "stepwise addition algorithm" which was introduced by J. Felsenstein [2] and implemented with some modifications in **fastDNAml** [7]. Furthermore, we will shortly outline the parallel algorithm of **parallel fastDNAml** and **PAxML**.

The calculation of the optimal phylogenetic tree for a set of rRNA input sequences based on the maximum likelihood method is NP-complete, due to the exponential growth in the number of possible tree topologies (e.g. there exist over 2 million possible topologies for 10 sequences). Thus, heuristics have to be introduced, in order to reduce the search space, i.e. the number of evaluated tree topologies. Suppose we have a set of n input sequences. A phylogenetic tree is an unrooted binary tree with the sequences at its leaves and with $2n-3$ inner nodes (each node of the tree has either degree 3 or degree 1).

The sequential algorithm, works as follows:

Suppose we have found the best tree t_k of size k, i.e. with k sequences at its leaves, according to the heuristics. Sequence $k+1$ consisting of a new branch with a new inner node and the sequence at either end, is then inserted into all $2k-3$ branches of t_k and the likelihood of the in that manner generated topologies $t_{k+1,1}, ..., t_{k+1,s}, s = 2k-3$ is calculated. After this step, local and/or global rearrangements of the best tree drawn from the set $t_{k+1,1}, ..., t_{k+1,s}$ are performed and evaluated, if the respective program option is set, in order to further improve the quality of the tree. The tree with the best likelihood of size $k+1$ is then used for insertion of sequence $k+2$. We call all tree topologies of size k, that are evaluated by the algorithm: "topology class of size k".

The algorithm starts with the only possible tree topology of size 3 using the first 3 sequences of the input data set and subsequently adds the remaining sequences as described above.

Since the most cost intensive part of the computation is the calculation of the likelihood value for each tree topology analyzed ($\approx 95\%$ of total computation time in the sequential program), the parallelization is straight-forward.

The parallel algorithm consists of a master, which is responsible for initialization, distribution of the input data, generation of tree topologies and gathering results.

The worker component simply performs the evaluation of a specific tree topology obtained by the master, i.e. computes its likelihood value. The topology to be evaluated is transformed into a simple, relatively short, string representation by the master and sent to a worker. Thus, especially since topology evaluation times increase with tree size $k, k = 4, ..., n$, the communication overhead is neglectible for the calculation of huge phylogenetic trees and the problem is well-suited for distributed computation (see Figure 4).

In **parallel fastDNAml** an additional foreman component has been inserted between master and workers for error-handling.

4 Implementation

For designing **DAxML** we initially simplified **PAxML** by removing the foreman component entirely from the system, since error handling can more easily be handled directly by **LMC**.

Furthermore, we changed the program structure, such as to create all tasks of size k, i.e. all topologies with k leaves, that can be evaluated independently

at once, and queue them in their string representation. This transformation was performed, in order to provide a means for issuing simultaneous topology evaluation requests (see below). Note, that several sets of trees of topology class

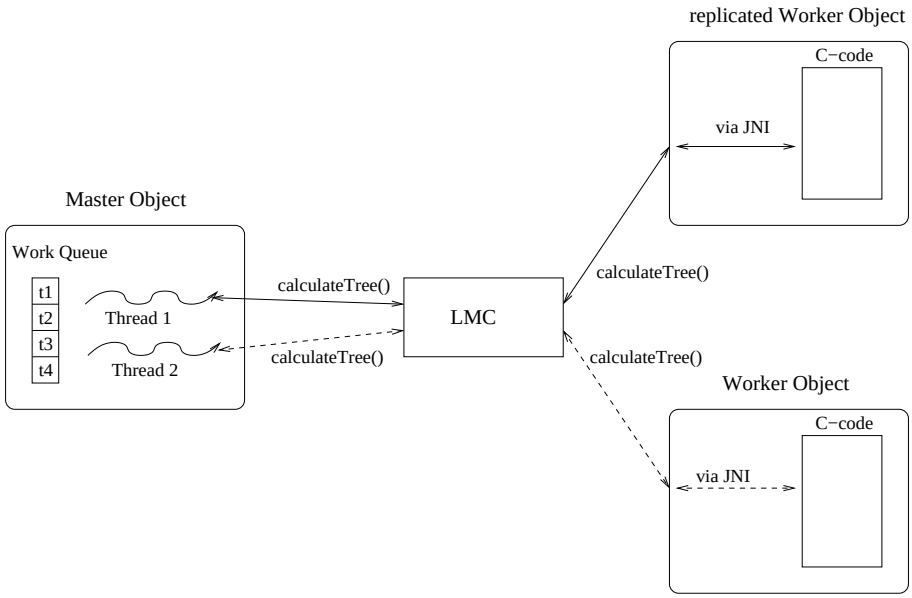

Fig. 2. System architecture of DAxML

k, that have to be evaluated in sequential order, may be generated, depending on the selected program options of **DAxML**.

Those sets are sufficiently large, such that they do not create a synchronization problem at the respective transition points.

The overhead induced by first creating and storing *all* topologies before invoking the evaluation function is neglectible, since the invocation of the topology evaluation function consumes by far the greatest portion of execution time.

Because **LMC** is based on a modified **JacORB** [1] version and only provides services for JAVA/CORBA applications, we initially transformed the simplified code into a sequential JAVA program using JNI (JAVA Native Interface). We designed two JAVA classes `Master` and `Worker` providing analogous functionalities as their counterparts in **PAxML**. The basic service provided by the `Worker` class is a method called `calculateTree()`, for evaluating a specific tree topology, which in turn invokes the fast native C evaluation function via JNI.

The `Master` component loads and parses the sequence file, passes the input data to the `Worker`, generates tree topologies and gathers results.

The transformation of the sequential JAVA code into a **LMC**-based application was straight-forward, since its class layout already complied with the structure of the distributed application. The `Worker` class is encapsulated as CORBA worker object, and provides its topology evaluation function as CORBA service. The state of the CORBA `Worker` object consists only of the sequence data, which can be loaded via NFS or directly from the `Master` when the `Worker` object is created either by initial placement, migration or replication.

Thus, since the sequence data is not modified during tree calculation, replications and migrations of worker objects do not induce any consistency problems.

In the main work-loop of the `Master`, a number of threads corresponding to the number of available hosts controlled by **LMC** is created, in order to perform simultaneous topology evaluation requests. This enables **LMC** to correctly distribute tree evaluation requests among worker objects on distinct hosts and to ensure optimal distribution granularity. The system architecture of **DAxML** is outlined in Figure 2 for a simple configuration with two worker objects.

5 Results

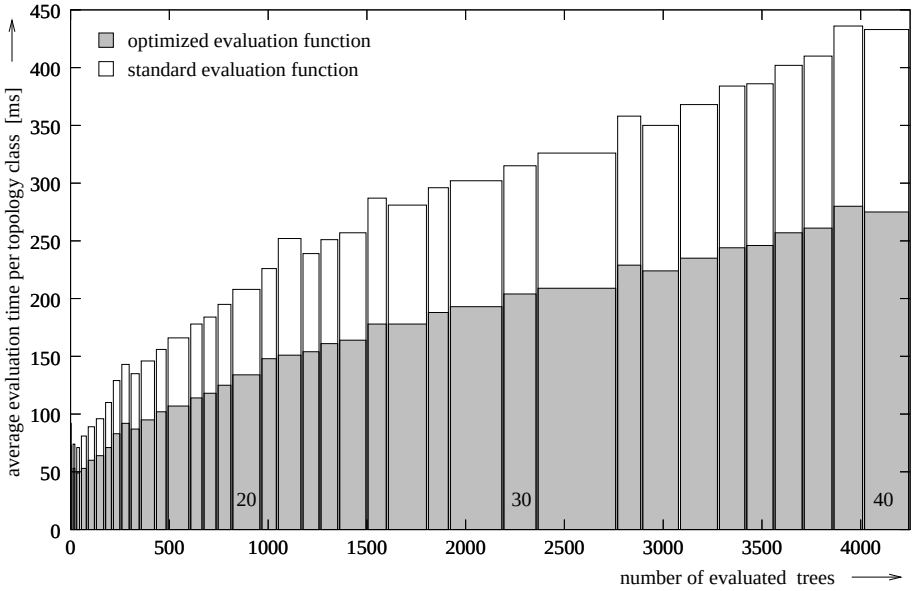

Fig. 3. Average evaluation time improvement per topology class: DAxML vs. parallel fastDNAml evaluation function

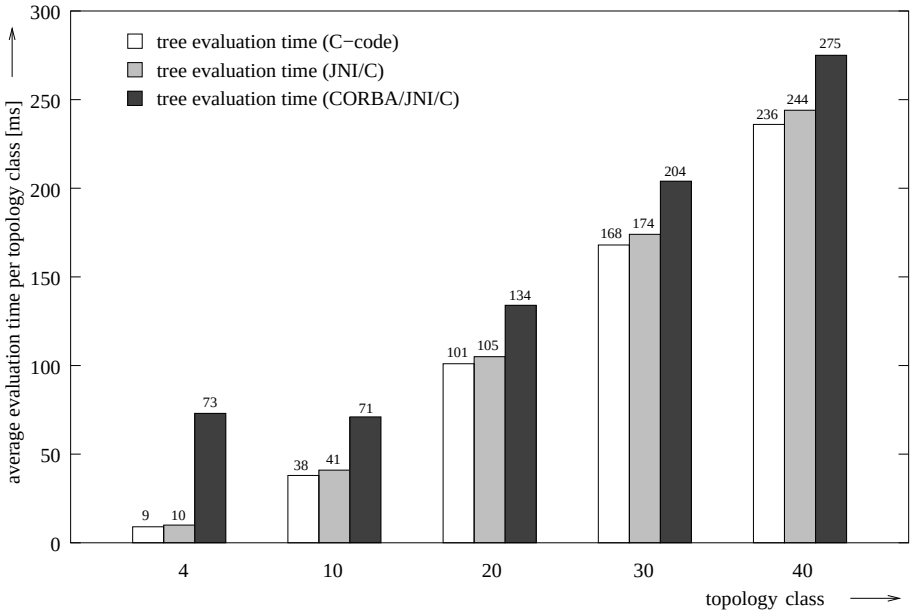

Fig. 4. JNI and CORBA-communication overhead

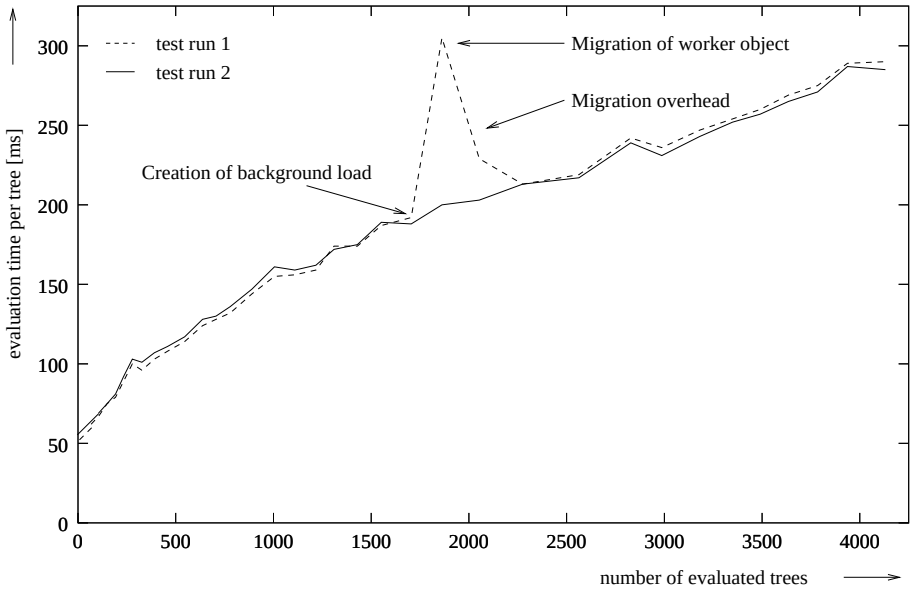

Fig. 5. Worker object migration after creation of background load on its host

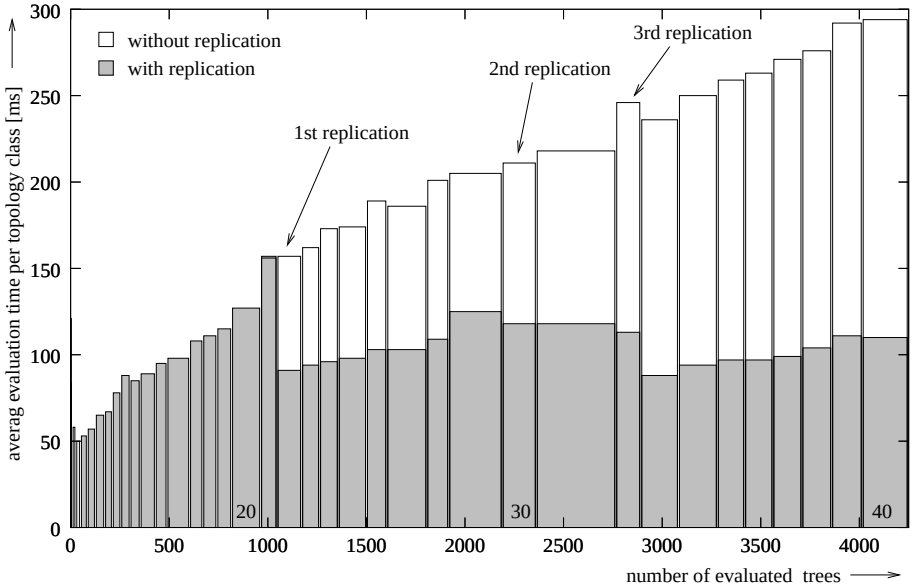

Fig. 6. Impact of 3 subsequent automatic worker object replications

We conducted performance analysis tests on 4 Ethernet connected Sun-Blade-1000 machines of the SUN cluster at the LRR using sufficiently large test sets of 20, 30, 40 and 50 sequences, extracted from the ARB database, in order to evaluate the behavior of **DAxML** and **LMC** in terms of CORBA/JNI overhead, impact of the algorithmic optimizations, and automatic worker object replication/migration.

In Figure 3 we demonstrate the impact of the algorithmic optimizations on the speed of the tree evaluation function including JNI and CORBA overhead. We conducted two **DAxML** test runs with a single worker object, using the standard and optimized tree evaluation function and measured the average tree evaluation time per topology class (see section 3) for a test set of 40 sequences. The algorithmic optimizations show analogous performance improvements as measured for the parallel and sequential program [11]. All subsequent tests were performed using our novel optimized evaluation function.

Another important aspect is the overhead induced by the integration of CORBA and JNI into **DAxML**. As previously mentioned the communication overhead decreases with increasing tree size, due to the fact, that average evaluation time per tree increases during the computation as depicted in Figure 3, whereas the amount of communicated data per topology class remains practically constant. For the same reasons and despite the fact, that we have used some heavy-weight JNI mechanisms such as JAVA callbacks from C, the JNI

overhead becomes neglectible as the tree grows, since only small amounts of data are passed through JNI.

We measured average C, JNI, and CORBA tree evaluation times for selected topology classes of size 4, 10, 20, 30 and 40. As can be seen in Figure 4 during the initial phase of the computation, i.e. for size 4 and 10, the CORBA overhead is relatively high but decreases significantly for increasing topology size.

In order to demonstrate the efficiency and soundness of **LMC** we performed test runs using worker object replication and migration.

Figure 5 depicts the correct response of **LMC** to an increase of background load on a worker object host. We performed two test runs with 40 sequences and a single worker object, i.e. the replication mechanism was switched off, located on the same initially unloaded node and measured the evaluation time per topology.

Around the evaluation of the 1750th tree topology during the first test run we produced external load on the worker object host, which lead to a significant increase in topology evaluation time. The unfavorable situation is correctly resolved by the load balancer and a migration of the worker object to an unloaded host is performed.

Finally, Figure 6 demonstrates how the average evaluation time per topology class is progressively being improved by 3 subsequent automatic worker object replications performed by **LMC**, compared to a run with automatic replication switched off.

6 Future Work

Current work focuses mainly on building a seti@home-like [9] distributed phylogeny program based on the `http` protocol and the novel randomized/distributed tree inference algorithm described in [13]. A parallel MPI-based prototype is already being evaluated. Within this context we plan to run big distributed phylogenetic tree calculations with data sets from the ARB database, using the available resources at the TUM. Furthermore, we will work on further improving our randomized tree inference algorithm, by extracting additional information from the set of trees calculated during the initial phase of the algorithm. Within this context we have already integrated the consensus tree program CONSENSE [4] into our parallel prototype.

References

1. Brose, G.: JacORB: Implementation and Design of a Java ORB. International Conference on Distributed Applications and Interoperable Systems (DAIS'97). Chapman & Hal (1997)
2. Felsenstein, J.: Evolutionary trees from DNA sequences: A maximum likelihood approach. J. Mol. Evol., Vol. 17. (1981) 368-376
3. Korber, B., Muldoon, M., Theiler, J., Gao, F., Gupta, R., Lapedes, A., Hahn, B.H., Wolinsky, S., Bhattacharya, T.: Timing the ancestor of the HIV-1 pandemic strains. Science, Vol. 288. (2000) 1789-1796

4. Jermiin, L.S., Olsen, G.J., Mengersen, K.L., Easteal, S.: Majority-rule consensus of phylogenetic trees obtained by maximum-likelihood analysis. Mol. Biol. Evol., Vol. 14. (1997) 1297-1302
5. Lindermeier, M.: Load Management for Distributed Object-Oriented Environments. Proceedings of 2nd International Symposium on Distributed Objects and Applications (DOA'00). IEEE Computer Society, (2000) 59-68
6. Lindermeier, M.: Ein Konzept zur Lastverwaltung in verteilten objektorientierten Systemen (A concept for load managment in distributed object-oriented systems). Ph.D. thesis. Technical University of Munich (2002)
7. Olsen, G.J., Matsuda, H., Hagstrom, R., Overbeek, R.: fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. Comput. Appl. Biosci., Vol. 10. (1994) 41-48
8. **ParBaum** homepage, **PAxML** download: `http://wwwbode.in.tum.de/~stamatak/research.html`
9. Search for Extraterrestrial Intelligence at Home: `http://setiathome.ssl.berkeley.edu/`
10. Stamatakis, A.P., Ludwig, T., Meier, H., Wolf, M.J.: **AxML**: A Fast Program for Sequential and Parallel Phylogenetic Tree Calculations Based on the Maximum Likelihood Method. Proceedings of 1st IEEE Computer Society Bioinformatics Conference (CSB 2002). IEEE Computer Society (2002)
11. Stamatakis, A.P., Ludwig, T., Meier, H., Wolf, M.J.: Accelerating Parallel Maximum Likelihood-based Phylogenetic Tree Computations using Subtree Equality Vectors. Proceedings of Supercomputing Conference (SC2002). IEEE Computer Society (2002)
12. Stamatakis, A.P., Ludwig, T., Meier, H.: Adapting **PAxML** to the Hitachi SR8000-F1 Supercomputer. Proceedings of 1. Joint HLRB and KONWIHR Workshop. (2002)
13. Stamatakis, A.P., Ludwig, T.: Phylogenetic Tree Inference on PC Architectures with AxML/PAxML. Proceedings of IPDPS2003, High Performance Computational Biology Workshop (HICOMB). IEEE Computer Society (2003)
14. Stewart, C.A., Hart, D., Berry, D.K., Olsen, G.J., Wernert, E., Fischer, W.: Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference. Proceedings of Supercomputing Conference 2001 (SC2001). IEEE Computer Society (2001)
15. Stewart, C.A., Tan, T.W., Buchhorn, M., Hart, D., Berry, D., Zhang, L., Wernert, E., Sakharkar, M., Fisher, W., McMullen, D.: Evolutionary biology and computational grids. IBM CASCON 1999 Computational Biology Workshop: Software Tools for Computational Biology. (1999)
16. The ARB project: `http://www.arb-home.de`