

Distributed High Performance Computing in Heterogeneous Environments with DOTS

Wolfgang Blochinger
University of Tübingen
Symbolic Computation Group
D-72076 Tübingen, Germany
blochinger@informatik.uni-tuebingen.de

Abstract

This paper deals with high performance computing in heterogeneous clusters using the Distributed Object-Oriented Threads System (DOTS). DOTS is a parallelization platform that enables the programmer to build distributed parallel applications using a high level and easy-to-use parallel programming paradigm. It integrates a wide range of different systems (from realtime kernels to mainframes) into a single system environment for high performance computing. A special design focus of DOTS is to optimize usability aspects.

Additionally, the paper discusses three advanced features of DOTS that especially support its deployment in heterogeneous environments for high performance computing. For defining the parallel environment and for documenting and evaluating the results of program runs an integrated XML centric approach is presented. A light-weight checkpointing mechanism which is seamlessly integrated into DOTS and especially designed for heterogeneous environments is described. DOTS is based on a highly customizable communication kernel. Run time measurements are presented that prove the efficiency of the communication kernel in heterogeneous clusters. Also, performance measurements of a non-trivial parallel application from the field of Symbolic Computation are presented, which revealed good speedups on a heterogeneous cluster of workstations.

1. Introduction

This paper deals with high performance computing in heterogeneous cluster system environments employing the Distributed Object-Oriented Threads System (DOTS) [4]. DOTS is a parallelization platform that integrates a wide range of different computing platforms into a single system environment for high performance computing. DOTS ad-

resses various aspects of heterogeneity like different hardware architectures, operating systems and different performance levels of the nodes in a cluster.

Although DOTS was originally designed as a software platform for the parallelization of algorithms from the realm of Symbolic Computation [3], it is also used in other application domains like parallel computer graphics [11].

One of the major design goals of DOTS is to achieve a high degree of overall usability. Using DOTS, tasks like system installation, as well as application programming, testing, evaluation and documentation of results can be carried out easily. This makes DOTS ready-to-use in existing parallel environments, for example pools of workstations.

The paper is organized as follows: After an introduction to the basic concepts of DOTS in Section 2, Section 3 discusses different aspects of the design of DOTS that are specifically related to the support of parallel distributed computing in heterogeneous environments. In Section 4 performance results of a distributed parallel application from the field of Symbolic Computation, obtained in a heterogeneous cluster of workstations are presented. The paper concludes with a comparison of DOTS with related systems that also support high performance computing in heterogeneous environments in Section 5.

2. Overview of DOTS

In this section an introduction to the basic concepts of DOTS is given. After a description of the programming model, the architecture of DOTS is outlined. The section concludes with a discussion of implementation issues.

2.1. Programming Paradigm of DOTS

DOTS provides object-oriented, asynchronous remote procedure call services for programs written in C++. These

services are accessed by the programmer through a lean threads-style API.

The main idea of the programming model of DOTS is to make the threads programming paradigm used on shared memory machines available in a distributed memory environment.

This approach for programming distributed memory architectures has several benefits:

- A hierarchical multiprocessor, consisting of a cluster of shared memory multiprocessor systems can be efficiently programmed using a single paradigm.
- It releases the programmer from performing a paradigm shift when transforming multithreaded shared memory applications into distributed parallel applications (or vice versa).
- Compared to the widely used paradigm of message passing, the threads paradigm better supports the creation of structured parallel programs (see the discussion in Section 5.1). This helps to improve the software quality of distributed memory applications.
- Programming with the threads paradigm shields the programmer from low level system details like explicit communication and data format conversions.

Figure 1 shows a distributed environment with three nodes. On node B two additional DOTS threads are created that are mapped for parallel execution on node A and on node C.

The DOTS API provides the following basic primitives:

- DOTS thread creation (`dots_fork`). An argument object for the created DOTS thread has to be supplied to `dots_fork`.
- Synchronization with the completion of the execution of another DOTS thread (joining) (`dots_join`). The computed result object of the joined DOTS thread is returned by `dots_join`.
- DOTS thread cancellation (`dots_cancel`).

All primitives can also be used in conjunction with so called *thread groups*. Thread groups are a means of representing related DOTS threads. When applied with thread groups, the semantics of each primitive is automatically changed to the appropriate group semantics. E.g. when using the join primitive with a thread group, *join-any* semantics will be applied, i.e. the first DOTS thread of the group that has completed its computation is joined. If no thread of the group has already completed the calling DOTS thread is blocked until a DOTS thread of the group finishes its execution.

Since each DOTS thread also can create other DOTS threads, not only simple types of parallel algorithms (like master-slave), but also more elaborate algorithm types (e.g. divide-and-conquer) can easily be realized.

2.2. The Execution of DOTS Threads

When a DOTS thread is created with the `dots_fork` primitive, a so called *thread object* is instantiated that represents the DOTS thread within the system during the complete execution process. It holds all information that is necessary to execute the DOTS thread, like the argument object, a system wide thread ID or the current execution state.

DOTS threads are executed under the control of the Execution Unit (see Figure 2).

It contains a thread queue in which newly created thread objects are enqueued. A pool of (OS native) worker threads dequeue thread objects from the queue and execute the corresponding DOTS threads. The number of worker threads can be determined by the programmer. Normally, the number of locally available processors is chosen.

After the execution of a DOTS is completed, its thread object is placed in one of the ready queues. Ready queues are dynamically allocated when a new thread group (see Section 2.1) is created. They contain the thread objects of those DOTS threads that are represented by the corresponding thread group.

To support the execution of DOTS threads in a distributed environment, the DOTS architecture includes additional components (see Figure 3).

- The Thread Transfer Unit transfers thread objects between the queues of execution units residing on different nodes. Before the transmission over the network the thread object is serialized. On the destination node the thread object is re-created out of its serialized representation and enqueued in the local thread queue or in one of the local ready queues, depending on its execution state.
- The Load Monitoring Framework traces all events concerning the execution of DOTS threads and provides status information like the current load or the current length of the thread queue. Based on the Load Monitoring Framework, different load distribution strategies can be implemented. Basically, the load distribution strategy triggers the transfer of thread objects and selects destination nodes.

In addition to the previously described components that are located on each node, some central components of DOTS are grouped into the so called DOTS Commander (see Figure 4).

The DOTS Commander program includes the following components:

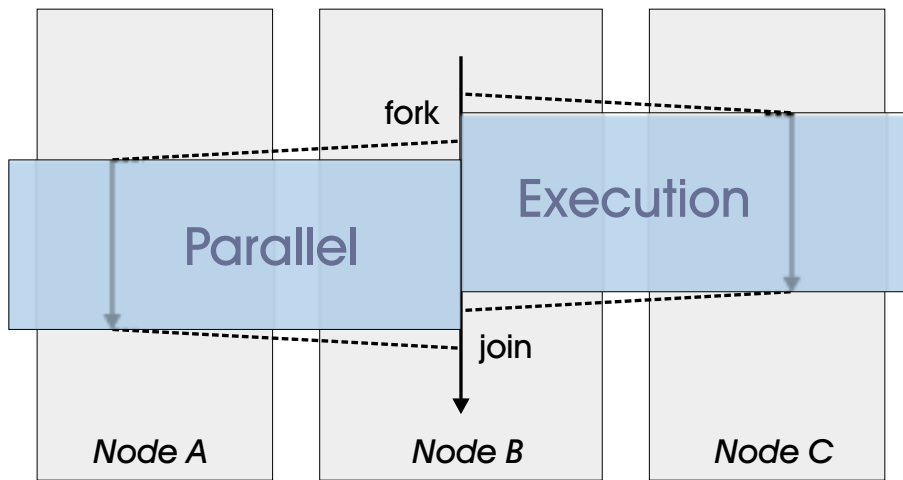


Figure 1. The DOTS programming paradigm

- The Session Unit controls the nodes by starting and shutting down the node processes and by distributing configuration information.
- The Logging Unit processes the logging records produced by the nodes.
- The Node Directory holds information about all involved nodes, e.g. the node addresses. If a central load distribution scheme is employed, the Node Directory can also hold information about the current load of each node.

2.3. Load Distribution

As described in the previous section, by using the Load Monitoring Framework different load distribution strategies can be integrated into DOTS. Currently, a centrally coordinated load distribution scheme and distributed load distribution schemes are implemented.

The centrally coordinated load distribution scheme uses centrally gathered load information to choose the execution node of DOTS threads.

The distributed schemes are based on work-sharing or work-stealing. The target resp. victim node can be selected randomly or by an application specific strategy. E.g. for master-slave algorithms work-stealing with a fixed victim node (the master node) can be used.

Using the Load Monitoring Framework, other load distribution schemes can easily be realized and integrated into DOTS.

2.4. Implementation

The implementation of DOTS is completely based on the Adaptive Communication Environment (ACE) [1]. ACE is a system-independent, object-oriented platform for developing high performance communication software. It is organized in a layered structure.

DOTS is based on the OS adaptation layer of ACE, which is a thin layer that homogenizes the APIs of the supported platforms into a system independent API.

ACE is available on an extensive range of systems, see [1] for a listing of the currently supported platforms.

3. Support for Heterogeneity in DOTS

DOTS supports a wide range of hardware and software platforms. Currently, DOTS is tested on the following platforms: QNX Realtime Platform, Windows 98/NT/2000, Solaris (Sparc/Intel), IRIX, AIX, FreeBSD, Linux and IBM OS/390 (In [2] a detailed description of DOTS running on clusters of IBM S/390 mainframes is given). As described in Section 2.4 DOTS is based on the ACE toolkit, so DOTS can easily be ported to other system platforms that are supported by ACE.

The following subsections treat some advanced features of DOTS that additionally support its deployment in heterogeneous environments for high performance computing.

3.1. High Level Programming Paradigm

Due to the high level parallel programming paradigm provided by DOTS, the programmer can be completely shielded from low level and platform specific system details

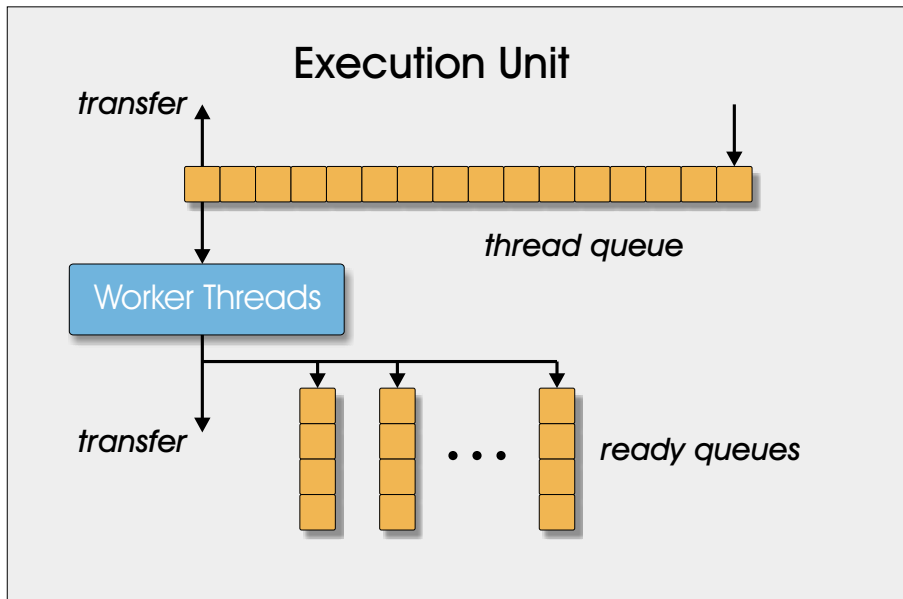


Figure 2. The DOTS Execution Unit

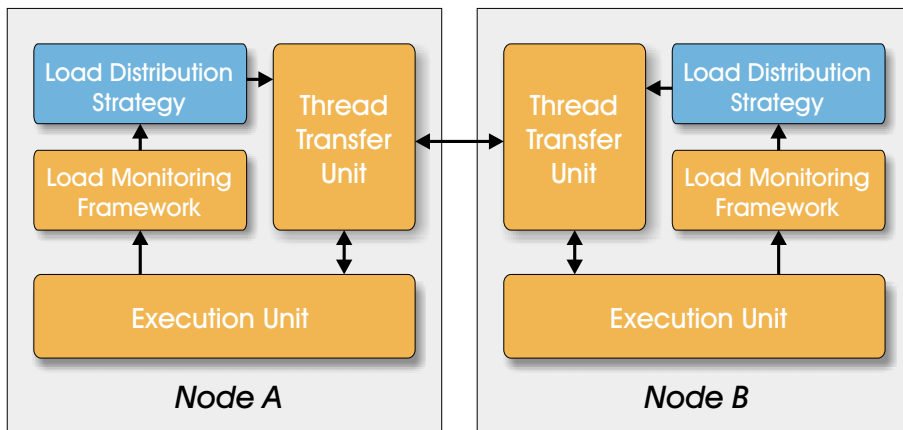


Figure 3. The DOTS Node Components

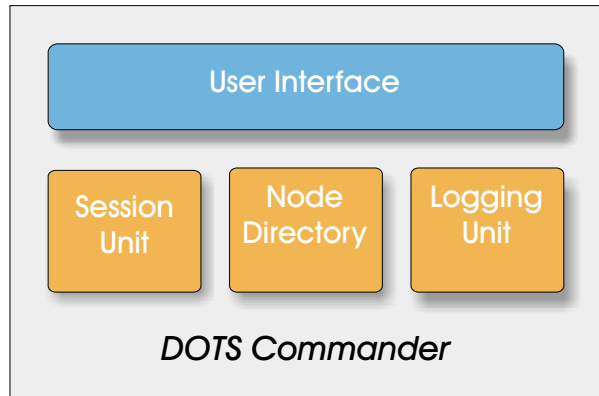


Figure 4. The DOTS Commander

like message passing, data conversions and synchronization issues. Using the high level fork/join parallel programming paradigm is the foundation of the support for heterogeneity in DOTS.

3.2. Dependability

Despite parallelization we often have to deal with extreme long running times of parallel applications. In this context fault tolerance is becoming an important issue. Fault tolerance is accomplished in DOTS with a light-weight checkpointing mechanism that particularly addresses the requirements of high performance computing in heterogeneous environments.

A typical solution for implementing checkpointing is to transparently save complete process images in predefined intervals. While being convenient from a programmer's point of view, this approach to checkpointing is not viable in a heterogeneous environment. With this method, a restart using a previously saved process image can only be carried out on a node that is of the same kind as the failed node. (Consider for example a process image taken as checkpoint on a Windows machine. It is clear that this image cannot be used to restart the process on a UNIX machine.) This fact limits the feasibility and also the efficiency of the prementioned approach to checkpointing considerably when used in heterogeneous clusters.

In order to avoid these drawbacks, DOTS pursues a different approach. It provides a lean checkpointing mechanism that can be explicitly controlled by the programmer. The DOTS API is extended with primitives to facilitate explicit programmer controlled checkpointing. The programmer specifies so called *checkpointing objects* that contain all application specific state information needed to perform a restart. After registering these checkpointing objects with the DOTS environment, an API call can be used to trig-

ger the checkpointing process at appropriate intervals. All checkpointing objects are saved in a platform independent serialized representation on stable storage. In case of a restart, the checkpointing objects are transparently initialized with the most recent checkpointed information.

3.3. Specification and Documentation of Computations

Typically, a heterogeneous cluster consists of a potentially large number of different kinds of nodes with varying configurations. Especially in larger clusters, the exact definition of the configuration of all nodes is a complex task.

DOTS uses an XML (Extensible Markup Language) [18] centric approach to determine the components of the cluster and to relate results of computations with an exact description of the environment in which they were obtained. Computations carried out with DOTS can be completely defined, documented and evaluated using a single XML document. A special DTD (Document Type Declaration) called DOTSMML has been defined to ensure that all XML documents describing DOTS computations conform to a uniform format. Figure 5 shows a slightly simplified version of the DOTSMML DTD.

Basically, DOTSMML documents consist of one configuration section and of several computation sections. Within the configuration section all aspects of the system environment are exactly defined. It includes detailed information about the integrated nodes and specifies all other system configuration parameters. Each computation section represents an individual computation carried out in the specified environment. Input data and computed results as well as other relevant data like measured running times and logging output can be included.

Using XML for the specification and documentation of computations in heterogeneous environments has several

```

<!ELEMENT dots (configuration, computation*)>
<!ELEMENT configuration (global, commander, node+)>
<!ELEMENT global EMPTY>
<!ATTLIST global
  archsize CDATA "1024"
  archdelta CDATA "1024"
  lifetime CDATA "24"
  strategy CDATA "1"
>
<!ELEMENT commander EMPTY>
<!ATTLIST commander
  rbufsize CDATA "-1"
  sbufsize CDATA "-1"
  backlog CDATA "16"
>
<!ELEMENT node (exec)>
<!ATTLIST node
  name CDATA #REQUIRED
  cpu CDATA #IMPLIED
  speed CDATA #IMPLIED
  ram CDATA #IMPLIED
  nic CDATA #IMPLIED
  os CDATA #IMPLIED
  maxload CDATA "1"
  rbufsize CDATA "-1"
  sbufsize CDATA "-1"
  backlog CDATA "16"
>
<!ELEMENT exec (#PCDATA)>
<!ELEMENT computation (input?, output?, time?, logging?)*>
<!ATTLIST computation
  name CDATA #IMPLIED
  date CDATA #IMPLIED
>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT logging (log)*>
<!ELEMENT log (#PCDATA)>
<!ATTLIST log
  node CDATA #IMPLIED
  level CDATA #IMPLIED
>

```

Figure 5. The DOTSMML Document Type Declaration.

advantages. XML is a standardized, platform-independent format, and is therefore eminently suited to be used in heterogeneous environments. It provides a well structured way of defining and representing all aspects of large and complex heterogeneous cluster environments along with detailed descriptions of computations that were carried out. Standard XML tools can be used to easily create, maintain, store and exchange DOTSMML documents. Related technologies like XSL Transformations (XSLT) [20] and XPath (XML Path Language) [19] can be used to automatically evaluate the results of DOTS computations or to transform DOTSMML documents to other representations, e.g. HTML. Figure 6 summarizes the integration of XML and DOTS

3.4. Communication Kernel

The (remote) execution of DOTS threads requires the transmission of thread objects as well as other system control messages between the nodes that comprise the parallel environment.

Since one major design goal of DOTS was to support a wide range of standard platforms within a single parallel environment it is not feasible to employ special network hardware and protocols like Myrinet [13] or SCI [15] to ensure efficient data transmission. Instead, communication in DOTS is based on the TCP/IP protocol suite which is available on almost all relevant platforms.

On order to optimize TCP/IP performance for heterogeneous environments, DOTS is built on top of a highly configurable communication kernel that supports connection caching, connection endpoint configuration and efficient automatic data format conversions.

3.4.1 Connection Caching

Besides having high bandwidths, a crucial point in distributed high performance computing is to achieve low latency for data transmissions.

In order to reduce latency, the communication kernel of DOTS implements a caching mechanism for TCP connections. Connections are not closed when a data transmission is completed but are cached for later re-use.

Connection caching avoids overheads in connection establishment (TCP's three-way handshake protocol and slow start algorithm see [16]) and termination (exchanging four FIN segments) and thus noticeably reduces the latency for data transmission.

Performance measurements presented in Section 4.1 show that connection caching significantly decreases the overhead of executing DOTS threads.

3.4.2 Connection Endpoint Configuration

The DOTS communication kernel allows for each node to individually configure connection parameters like buffer sizes and backlog values for passive endpoints.

Potentially, a heterogeneous parallel environment consists of slower and faster machines (both in terms of CPU and network performance). In this case, the buffer sizes for connection endpoints should be adapted to minimize the synchronization overhead and consequently improve bulk data transmission between machines with different speeds. As a rule of thumb, fast machines should have large send buffers while slow machines should be configured with large receive buffers.

As shown in Section 4.1, determining optimal connection parameters for each node can decrease the execution overhead of coarse grained DOTS threads in heterogeneous environments considerably.

3.4.3 Data Format Conversions

In heterogeneous environments we have to deal with communication between hosts with different data representations (like different sizes of primitives datatypes, different byte orderings (big vs. little endian) or different character encodings (ASCII vs. EBCDIC)).

The common approach to handle communication between hosts with different data formats is to define a standard representation for transmission, e.g. "network byte order" or XDR (External Data Representation) [17]. While being feasible, this procedure can cause unnecessary conversion overhead, when two nodes with the same but not the standard transmission data representation exchange data. In this case there are two unnecessary conversions.

The DOTS communication kernel ensures that data conversions are only carried out when the involved nodes actually have different data representations. The sender always sends data in its native format along with a compact platform-independent description of the format. The receiver uses this information to trigger an appropriate conversion procedure if needed.

4. Performance Measurements

In this section performance measurements carried out in a heterogeneous environment consisting of 11 different nodes (cf. table 1) are presented. All nodes except node 11 were directly connected through an ethernet switch. Node 11 was integrated via the campus FDDI backbone.

All time values shown in the next sections are the arithmetic mean of wall clock times taken from three individual program runs.

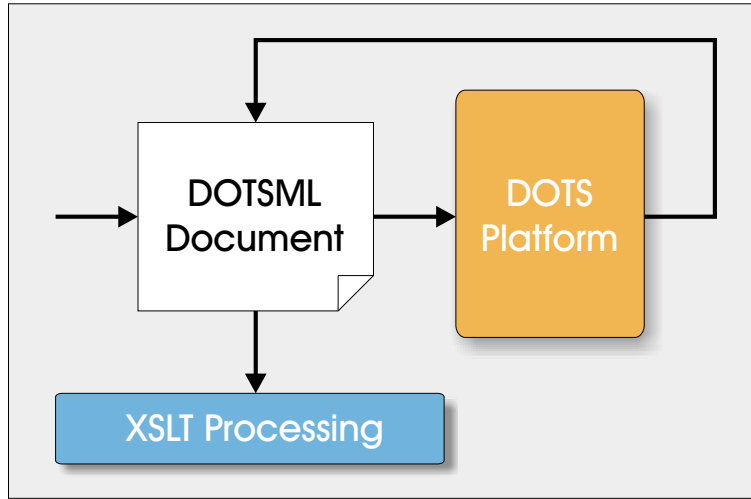


Figure 6. Using DOTSML Documents in DOTS

node	CPU (speed/MHz)	RAM (MByte)	NIC Speed (MBit/s)	Operating System
1	UltraSparcII (400)	2048	100	Solaris 7
2	PentiumII (300)	128	100	Solaris 7
3	PentiumII (400)	128	100	FreeBSD 4.1
4	PentiumII (400)	128	100	Linux (Kernel 2.4)
5	PowerPC (60)	32	10	AIX 4.3
6	UltraSparcI (140)	64	10	Solaris 7
7	PentiumIII (500)	256	100	Windows 2000
8	PentiumII (400)	128	100	Windows NT4
9	PentiumIII (600)	256	100	QNX 4
10	Pentium (233)	64	100	Windows 98
11	MIPS R8000 (100)	1024	100	IRIX 6.2

Table 1. Parallel environment used for performance measurements

4.1. Execution Overhead of DOTS Threads

In order to quantify the impact of the optimizations provided by the communication kernel on the overall DOTS performance, the thread execution overhead of DOTS threads for different granularities (i.e. different argument and result sizes) was measured. The thread execution overhead is defined as the time of executing a DOTS thread that does not carry out any computation. For the subsequently described experiments a central load distribution scheme was used. This mode of load distribution generally imposes the highest communication overhead for executing DOTS threads. Additionally, local execution on the node that created the DOTS threads was disabled. This means that for each DOTS thread the corresponding thread object had to be transmitted over the network two times.

In all measurements, 1000 DOTS Threads with different

argument and result sizes were executed in the described cluster. Based on the total run time, the average execution time for one thread was calculated.

A first series of measurements was made to analyze the impact of connection caching. Table 2 shows the average thread execution overhead with and without connection caching.

In a second series of measurements the effect of connection endpoint configuration was examined. Table 3 compares the average thread execution overhead using system default values with the overhead using individually tuned parameters.

4.2. Parallel Satisfiability Checking

As an example application from the field of Symbolic Computation, parallel satisfiability checking for boolean

Arg+Result Size (Bytes)	time(ms) (without cache)	time (ms) (with cache)	speedup
32	5.5	1.2	4.6
64	6.2	1.2	5.2
128	6.6	1.1	6.0
256	7.4	1.2	6.2
512	7.6	1.2	6.3
1024	6.9	1.2	5.8
2048	6.5	1.2	6.5
4096	6.1	1.2	5.5
8192	5.0	1.4	3.6
16384	4.6	2.2	2.1
32767	4.9	4.7	1.0

Table 2. Average thread execution overhead for different argument and result sizes with and without connection caching.

Arg+Result Size (kB)	time (ms) (default values)	time (ms) (tuned values)	speedup
64	10.9	8.9	1.2
128	23.3	17.7	1.3
256	50.7	33.9	1.5
512	112.4	87.7	1.3
1024	225.1	182.6	1.2

Table 3. Average thread execution overhead for different argument and result sizes with and without tuning of endpoint parameters.

formulae (SAT) is considered. The SAT problem asks whether or not a Boolean formula has a model. Important applications of the SAT algorithm include cryptography, planning and scheduling or hardware verification. The computational complexity of these problems can vary considerably, where the hardest practical problems can require thousands of hours of running time. So parallelization of satisfiability checking algorithms is an important issue.

For implementing a distributed parallel satisfiability checker with DOTS the parallelization techniques described in [21] were adopted.

As input, the benchmark *pret60_40* from the publicly available DIMACS benchmark suite¹ for satisfiability provers was chosen. Table 4 shows the parallel run time and the sequential run times for each node in the described environment.

Due to the lack of sufficient physical memory on the nodes 5, 6, and 10 the sequential run times (i.e. the run times to treat the whole problem) are disproportionately high. This effect does not appear in the parallel application, since each node only has to treat smaller subproblems at a time. For this reason speedup value are not calculated, it would not correctly reflect the actual situation.

¹[ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/](http://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/)

5. Related Work

In this section DOTS is compared to other systems that also support high performance computing in heterogeneous environments or that use a similar programming paradigm.

5.1. Message Passing Interface (MPI)

The Message Passing Interface (MPI) is a standard for message passing libraries defined by MPI forum [12]. It defines a set of primitives for point-to-point message passing communication and collective communication among parallel tasks. Implementations of MPI are available for different programming languages and for a wide range of system platforms.

On the one hand, parallel programs that use the message passing paradigm can be highly optimized. But on the other hand, they have a disposition to become unstructured. (Message passing is often called the assembly language of parallel programming.)

DOTS uses a higher level programming paradigm that enables structured programming and consequently leads to improved software quality. This also helps the programmer to concentrate on conceptual issues, instead of dealing

node	Sequential Times (s)	Parallel Time (s)
1	765	121
2	927	
3	1366	
4	879	
5	11431	
6	2089	
7	647	
8	779	
9	446	
10	4141	

Table 4. Run times for the sequential and the parallel satisfiability checking application

with low-level communication details. Additionally, DOTS provides a complete runtime environment with support for different load distribution strategies.

5.2. Globus

Globus [10, 8] is an extensive software infrastructure for the construction of persistent, complex computational grids [9] (metacomputers) from geographically distributed, heterogeneous computing resources. Rather than focusing on a particular programming model, it pursues the goal to provide paradigm-independent basic services, on top of which the developer can realize more specific parallel programming models or applications.

In contrast to Globus, DOTS is optimized for the distributed threads paradigm, both in terms of the supported primitives (like join-any or thread cancellation constructs) and the corresponding execution environment.

Although DOTS is also well suited to integrate heterogeneous environments into a homogeneous computing platform, it is not primarily targeted to be used for building complex geographically distributed computational grids. Instead, the design of DOTS mainly concentrates on usability aspects. The overall goal of DOTS is to minimize the time needed to install and maintain the parallel software infrastructure and the time needed to build, execute and evaluate parallel applications. It aims to provide a very lean and easy-to-use parallelization platform that enables application to be efficiently executed in an existing local heterogeneous parallel environment without further hardware or software requirements.

This approach makes DOTS particularly suitable to be used for example by scientist outside of computer science or for rapid prototyping of parallel algorithm layouts.

5.3. ProActive PDC

ProActive PDC [14, 5] (formerly Java/) is a system for parallel and distributed computing in Java, that is com-

pletely implemented in Java. It provides support for active-objects with asynchronous method calls and automatic future-based synchronization.

ProActive and DOTS both support the implementation of object-oriented parallel applications (written in Java or in C++, respectively).

The use of Java for high performance computing in heterogeneous environments has both advantages and disadvantages. Due to the platform neutral Bytecode representation of compiled Java programs, Java is eminently suited for heterogeneous environments. However, the interpretation process of Java Bytecode within a Java Virtual Machine produces a serious performance deterioration (compared to the execution of native machine code), which is clearly undesirable in the field of high performance computing ².

To avoid the aforementioned drawbacks, the DOTS runtime employs Java only in performance-insensitive components like GUIs or XML processing.

5.4. Cilk

Cilk [6] is a language for multithreaded parallel programming that represents a superset of ANSI C. It uses pre-compiling techniques for static code instrumentation in order to support the Cilk runtime system.

DOTS is library based and avoids therefore typical problems of systems that extend standard languages like the lack of standard development tools (e.g. debuggers). Moreover, DOTS is based on C++ and supports object-oriented programming.

There exists prototype implementation of a distributed version of Cilk, called Distributed Cilk [7] that spans clusters of SMPs. Distributed Cilk currently is only available

² Highly optimized Java Virtual Machines (using dynamic code analysis along with selective just-in-time compilation techniques) can increase the performance of Java applications considerably. But Java Virtual Machines providing these advanced capabilities are currently only available on a few platforms. This fact again limits the usability of Java in highly heterogeneous environments.

on a few platforms. Thus the usability of Distributed Cilk in heterogeneous environment is limited.

References

- [1] The ADAPTIVE Communication Environment (ACE). <http://www.cs.wustl.edu/~schmidt/ACE.html>. 2.4
- [2] W. Blochinger, R. Bündgen, and A. Heinemann. Dependable high performance computing on a parallel sysplex cluster. In H. R. Arabnia, editor, *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, volume 3. CSREA Press, 2000. 3
- [3] W. Blochinger, W. Küchlin, C. Ludwig, and A. Weber. An object-oriented platform for distributed high-performance symbolic computation. *Mathematics and Computers in Simulation*, 49:161–178, 1999. 1
- [4] W. Blochinger, W. Küchlin, and A. Weber. The distributed object-oriented threads system DOTS. In A. Ferreira, J. Rolim, H. Simon, and S.-H. Teng, editors, *Solving Irregularly Structured Problems in Parallel*, number 1457 in Lecture Notes in Computer Science, pages 206–217. Springer-Verlag, 1998. 1
- [5] D. Caromel, W. Klauser, and J. Vayssiere. Towards seamless computing and metacomputing in java. *Concurrency: Practice and Experience*, 10(11–13):1043–1061, 1998. 5.3
- [6] The Cilk Project. <http://supertech.lcs.mit.edu/cilk/>. 5.4
- [7] distributed Cilk. <http://supertech.lcs.mit.edu/cilk/home/distcilk5.1.html>. 5.4
- [8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Journal of Parallel and Distributed Computing*, 11(2):115–128, 1997. 5.2
- [9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999. 5.2
- [10] The Globus Project. <http://www.globus.org>. 5.2
- [11] M. Meißner, T. Hüttner, W. Blochinger, and A. Weber. Parallel direct volume rendering on PC networks. In *The proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, Las Vegas, NV, U.S.A., July 1998. 1
- [12] Message Passing Interface Forum. <http://www.mpi-forum.org>. 5.1
- [13] Myricom. <http://www.myri.com>. 3.4
- [14] ProActive PDC. <http://www-sop.inria.fr/oasis/ProActive/>. 5.3
- [15] SCI Association. <http://www.scizzl.com>. 3.4
- [16] W. R. Stevens. *TCP/IP Illustrated: the protocols*, volume 1. Addison-Wesley Publishing Company, 1994. 3.4.1
- [17] R. Strinivasan. XDR: External Data Representation Standard. RFC 1832. 3.4.3
- [18] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.0*, 1998. <http://www.w3.org/TR/REC-xml>. 3.3
- [19] World Wide Web Consortium (W3C). *XML Path Language (XPath) 1.0*, 1999. <http://www.w3.org/TR/xpath>. 3.3
- [20] World Wide Web Consortium (W3C). *XSL Transformations (XSLT) 1.0*, 1999. <http://www.w3.org/TR/xslt>. 3.3
- [21] H. Zhang, M. P. Bonacina, and J. Hsiang. PSATO: A distributed propositional prover and its application to quasi-group problems. *Journal of Symbolic Computation*, 21:543–560, 1996. 4.2