Performance Analysis of Distributed Applications with ANSAmon[*]

B. Meyer[a], M. Heineken and C. Popien

[a]Aachen University of Technology, Computer Science IV,
Ahornstr. 55, 52056 Aachen, Germany, e-mail: bernd@i4.informatik.rwth-aachen.de

Distributed computing platforms like DCE or ANSAware offer a number of services to cope with distribution in application programming. With these services some aspects of distribution can be made transparent to programmers. In return programmers partially abandom control over performance of their application. To avoid performance bottlenecks, a distributed applications behaviour needs to be monitored. In this paper we present a distributed monitor called ANSAmon, that we developed for the ANSAware platform. It is realized according to a master-slave architecture and is a distributed application itself. Monitoring experiments can be configured in advance, whereas analysis is done after collecting all events. Global times will be estimated by a new method improving existing ones. Applying ANSAmon we investigated the ANSA REX protocol and the ANSA trader. Important conclusions for the efficient use of ANSAware for distributed applications can be deduced from the results obtained.

Keyword Codes: C.2.4; C.4;D.4.4
Keywords: Distributed Systems, Performance of Systems, Communications Management

1. INTRODUCTION

The ongoing migration from large mainframe computers to networks of workstations derives new requirements to application programmers. Whereas the former systems allow monolitic programs to be developed, modern systems are organized in a client-server fashion. As a consequence, a means for programming inter-workstation cooperation is necessary. The most common method is the remote procedure call (RPC), see [BiNe84]. Several distributed computing platforms offering appropriate services have been developed in the last years. Examples are the distributed computing environment (DCE) of the Open Software Foundation [DCE], the common object request broker architecture (CORBA) of the Object Management Group [OMG] or the ANSAware (Advanced Network Systems Architecture) of the AMP [ANSA]. Besides these commericial products, ISO developes a reference model for Open Distributed Processing (ODP) [ODP p2], [ODP p3]. It consists of concepts and their

[*] This work was partially supported by the Deutsche Forschungsgemeinschaft under Grant No. Sp 230/8-1

formal interpretation for distributed system design, general functions for building an infrastructure and means for bridging heterogenity between interfaces.

A lot of services and distribution transparency are offered by distributed computing platforms, that makes the developement of distributed applications more convinient. In return application programers partially abandom control over the performance of their application. Often, unconvenient use of offered services leads to enormous perfomance loss. This is partially caused e.g. by communication or syncronisation delay. First step is to find the performance bottlenecks before improvement can be realized. Therefore, performance characteristics have to be ascertained for subparts of an application and for infrastructure services. Performance evaluation of communication and computer system have been studied for a longer time, see [Jai91], [Kob78]. These techniques can be classified into analytic methods, simulations and measurement tools. First of all, a model of the system under study has to developed. Queueing systems and Petri Net models are common means for analytical tools and simulations, see [Kle75], [ICT93]. We developed an analytic tool for evaluating fork-join queueing nets, that supports performance management of service and applications presented in [MePo93], [PMS94], but we are not going into detail here. Whereas simulation tools are able to handle more general kinds of nets, analytical techniques are much faster in calculation. Often simulations still produce results, where analytic tools fail because of an state explosion of the corresponding models. Measurement can be further subdiveded into monitoring and benchmarking. In this work we will focus on monitoring, see also [ChSu92] for monitoring in network management systems.

The second section introduces the topic of distributed monitoring, especially focussing on differences to monitoring a single computer. A distributed monitor for ANSAware will be introduced in section three, whereas section four shows an example monitor session using ANSAmon and performance results obtained by monitoring the RPC protocol of ANSAware and its trader. A brief summary and fields for future work will conclude this article.

2. MONITORING OF DISTRIBUTED SYSTEMS

In the area of monitoring, an object system denotes the system to be mointored, whereas all components used for monitoring are named the monitoring system. These are logical concepts, that does not necessarily be physically separate, e.g. the instrumentation statements in an application program logically belong to the monitoring system, but physically to the object system. The complete monitoring process can be divided into several subtasks, see figure 2.1. First of all, events producted by the object system need to be made accessible to the monitor system. In general, there are hardware and software monitors. The difference depends on the kind of components used for monitoring. Hardware monitors employ special hardware devices for recording system events. Software monitors need event submitting statements to be introduced into application program code. This activity is called instrumentation of the application. Each event statement forwards a message to the monitor system containing event type and additional data in a special data structure called event record. This is called the recording of events. The monitor system checks incoming event records if they contain a relevant event type. All event records containing relevant event types are stored in a so-called event trace, which gives a temporal ordering to event records induced by their arrival time. After the recognition of events it is possible to filter certain events. Filtering checks if data in event records satisfy certain (filter) constraints, which are specific to each experiment. Even in central computers it is possible to have more than one event recording unit, e.g. if software and hardware monitoring is combined. Before starting analysis, event traces of all event recording components are collected. After collection, all monitoring data resides in a single place, where analysis takes place. In general it has to merge all event traces into a single temporal ordering. This requires a global time for all events [Lam78], which can be provided by a global time service or can be estimated after finishing an experiment. The result is usually stored in a program activity graph, that defines a partial order on events of a program. From this model, more complex metric values can be calculated. These data can be presented to human users by different kinds of graphical techniques like curves or diagrams. This completes to monitoring process. There have been several approaches proposed, see [LKG92], [JLS+87], [Mil89], but most of them are bound to a special kind of computer or operating system.

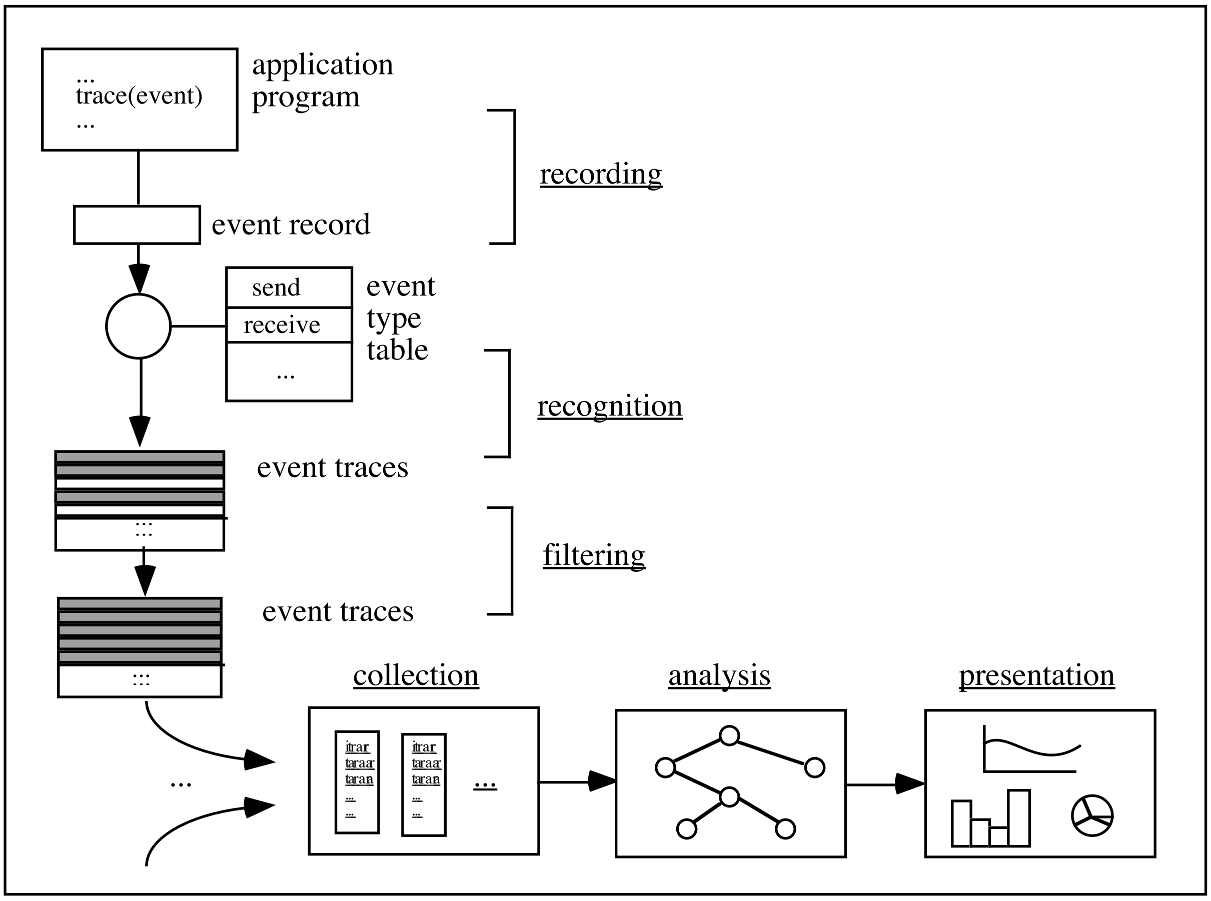

Figure 2.1. The general monitoring process

The monitoring results can be further interpreted and finally leading to instructions determining how to change the behaviour of the object system. In the best case, an improvement can be achieved. These activities are related to network and system management and commonly divided into functional areas like accounting, configuration, fault, performance and security management, see [OSI Man]. Before starting monitoring experiments, relevant performance metrics have to be determined. Target performance metrics for monitoring distributed applications can be e.g.

- the time spent in the communication system for executing remote procedure calls,
- the time spent for transport service,
- the throughput of the communication channel,
- the time spent in server waiting queues,
- the current length of server queue,
- critical paths of an application execution,
- the time spent with joining spawned threads.

All performance metrics can be useful as current or average values or as an value trace, that is combined with a time stamp, see also [Rol94].

3. CONCEPTS AND REALISATION OF ANSAMON

After a brief introduction to ANSAware we are going to describe concepts and design decisions made for the developement of ANSAmon.

3.1 Basic concepts of ANSAware

ANSAware is an infrastructure for developing and running distributed applications. It is available for a number of operating systems like SunOS, HP/UX, VMS and MS DOS. Communication between computer nodes is provided by socket interface of a transport services according to the TCP or UDP protocol. Heterogenity of programming language and operating system is bridged by a RPC mechanism and a common language for abstract interface description called the interface definition language (IDL).

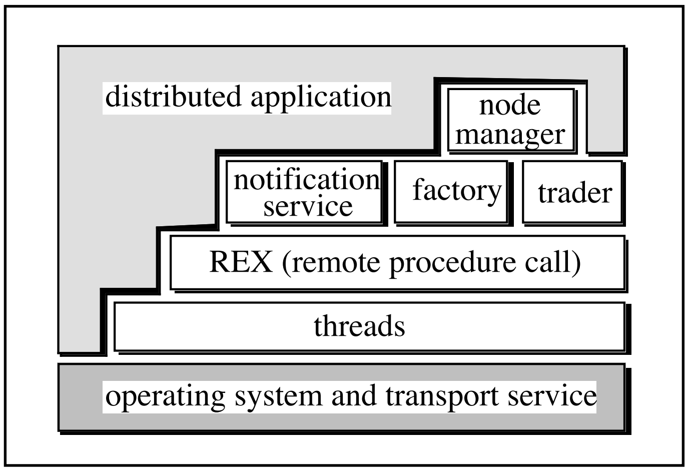

Figure 3.1. Global architecture of ANSAware

Besides RPC ANSAware provides a thread mechanism and three server types, see figure 3.1. A factory server is able to create server processes called capsules, and interfaces for a certain service type dynamicaly on the local node. References to these interfaces are mediated by a server called trader. It stores all public service offers and delivers appropriate references on request. A more comfortable way to deal with services and servers is provided by the node manager. It enables creating and deleting server on any node in the distributed system. In addition, cooperations between different trader server can be established.

3.2 The design of ANSAmon

Main goal for the development of ANSAmon is to achieve a minimal pertubation of the running application to be monitored. We decided to develope a software monitor, because we want the monitor system to be portable. Hardware monitors requires certain devices to be available for different kind of systems and vendors, but this assumption is not reasonable in our case. Although we used functions of the ANSAware platform, design has been done carefully to encapsulate ANSA-specific code in a few modules. Since ANSAware does not provide a global time service, we chose to estimate global timings. This choice was only possible, because we decided to perform analysis post-mortem, that means after finishing an experiment. It allows us to compute time-consuming performance characteristics as well as it contributes to the goal of interferring regular program execution as less as possible.

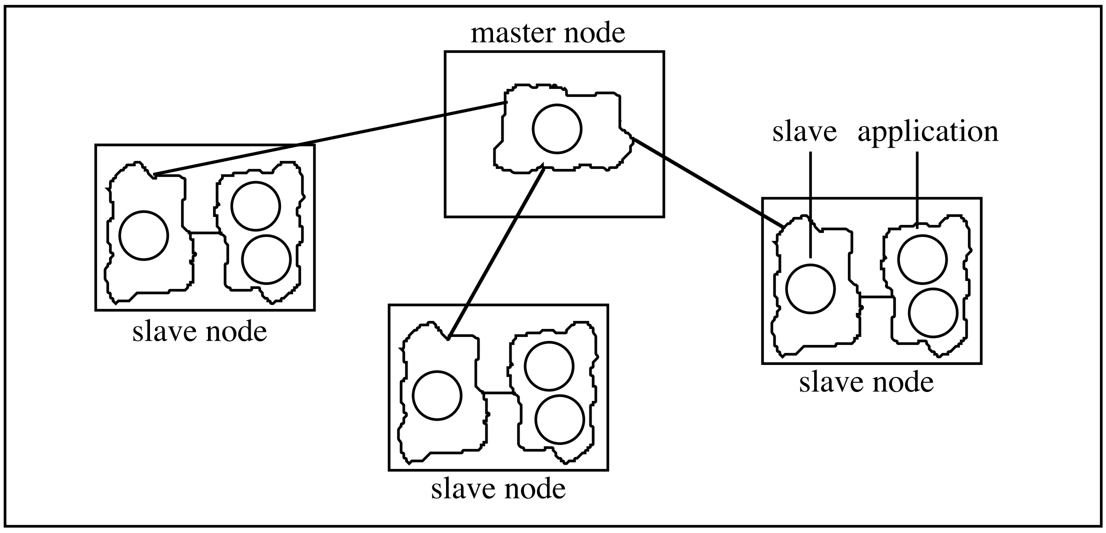

Figure 3.2. Global Architecture of ANSAmon

ANSAmon is designed as a distributed monitor with master-slave architecture, see figure 3.2. This choice contributes to the fact, that almost all of todays distributed computing is done in a client-server manner. Consistenly, ANSAware applications cooperate via RPC. A complete monitoring system consists of one master monitor (or master) and an arbitrary number of slave monitors (slaves). For each node participating in a distributed application there will be one slave. The master starts a monitor session at all its slaves. In order to have a minimal time for recording events and forwarding event records, each application manages two buffers. One is currently used for storing occuring events until it is full. Afterwards, the complete buffer is sent to the slave residing on the same node as the application, while the next events are store in the other buffer. The slaves collect incoming events from the application, stores and forward them to the master after the end of an experiment. While monitoring an application ANSAmon traces the occurence of certain events. These events can be predefined or defined by users. ANSAmon supports five kinds of predefined events, see table 3.1. These events occur while performing an RPC, as it is common in client-server-style programming.

Table 3.1. Predefined ANSAmon events

event type	description
send	transmission of data from client buffer, e.g. data for a RPC
receive	arrival of data in server buffer
dispatch	start of RPC operation at server
fork	spawning a new thread
join	waiting for a spawned thread to terminate

Within ANSAmon events are stored in a data structure called event record, see figure 3.3. It consists of an event type denoted by an identifier, some common event properties and some event-specific data. All events are described by a time stamp using local time and identifiers for monitored thread and capusule. For example an event record of a dispatch event additionally contains beside the event type the current number of threads in the server capsule. With this information the distribution of clients served for a certain server can be calculated for the monitored interval of time. Some analysis already can be done on local event traces, but this option is not supported yet.

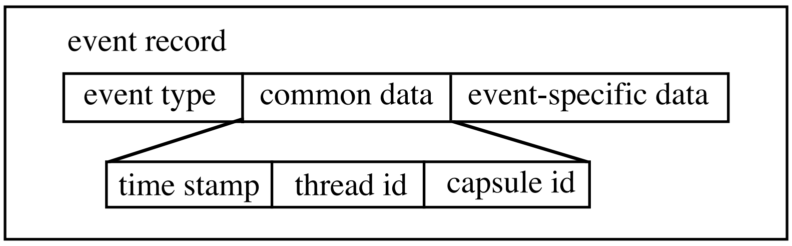

Figure 3.3. Structure of an event record

Before starting analysis, the master has to collect event traces of all slaves, which have to be merged. The basis for analysing traced events is a so-called program activity graph (PAG), see figure 3.4. It consists of a number of nodes representing traced events and directed arcs connecting two or more nodes giving the events a temporal ordering. Some events e.g. `fork` nodes have more than one outgoing arcs, whereas others e.g. `join` nodes can have more than one incoming arcs. A `send` node is always followed by at least one `receive` node. It represents an ordinary directed point-to-point communication. For the case of more than one successing `receive` nodes, a multicast communication is modelled. First, all event traces have to be merged and ordered. This requires finding a corresponding `receive` event for each `send` event. Before this can be done, all local time stamps have to be transformed to an estimated global time. Afterwards, the correspondence between `send` and `receive` events can be determined using global time stamps and an unique sequence number for each communication. Therefore it is assumed, that each message sending causes a reception on the other side and sending order is preserved. The sequence number is specific to `send` and `receive` events. Once a pair is detected, both nodes are replaced by a single node including the delay for communication.

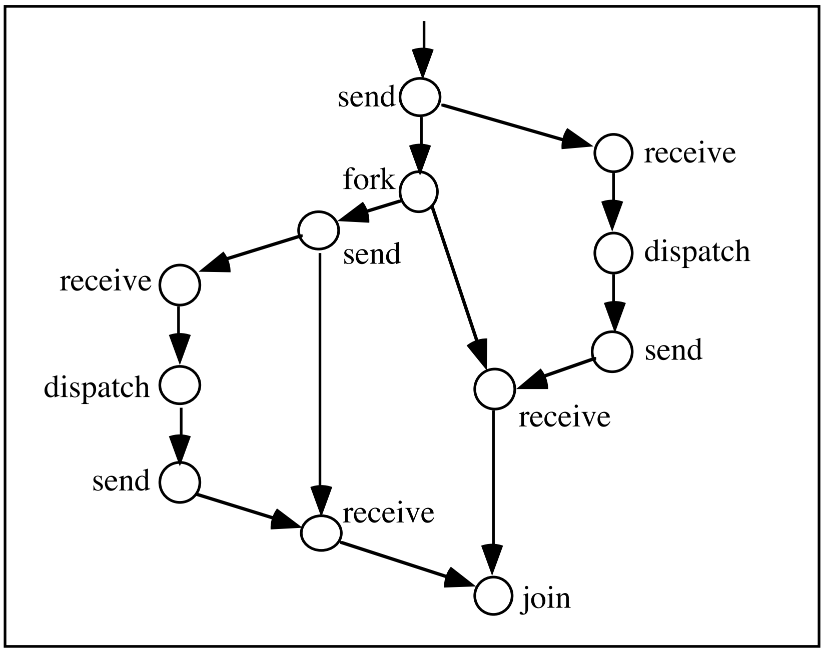

Figure 3.4. Example of a program activity graph

For estimating global time, an improved procedure for ANSAmon has been developed in [Hei94]. Like [DHH+87] it assumes the divergence between local times to develope in a linear manner. Global time estimation is consistently done by linear regression. Precision of global time approximation depends on the number of communication events occuring in an experiment. In contrast to [HKL+90] it does not only use extreme values for sending and receiving for computing regression parameter, but follows an adaptive approach using all available communication events.

4. MONITORING ANSAWARE USING ANSAMON

This section contains results obtained by sample monitor runs analysing the average response time of the ANSA RPC and of import requests to the ANSA trader. The underlying network for all experiements is a LAN.

4.1 Monitoring the ANSAware REX protocol for inter-capsule cooperation

First task of a RPC is to store a compressed version of an operation name and its parameters in a buffer. This is called marshalling, whereas the reverse operation is named unmarshalling. The first time a cooperation via RPC is set up between two capsules, an entry is added to both, channel and session table. Whereas the channel table holds information about all connections between capsules, the session table holds only information about currently active connections; that means, the entries of the session table are a subset of those of the channel table. The remote execution protocol (REX) handles the RPC. If the buffer is too large to be transported by a single packet of the underlying transport service, REX divides it into a number of appropriate fragments. These fragements are handed to the message passing system, that sends them using one of the offered transport services, e.g. TCP or UDP. All following experiments make use of UDP transport service. On the receiving side each incoming call is stored in a thread queue. Threads on the other side need a task to be processed. ANSAware performs non-preemptive scheduling for assigning free tasks to waiting threads. The RPC response time is calculated adding times of its most significant subtasks. It consists of the time for calling, performing a remote operation and returning results. In the following it is shown how the calling time is calculated within our experiments

time for call = marshalling + access channel table + access session table + REX + MPS + transport service delay+ MPS + REX + access session table + access channel table + unmarshalling + thread queue + task queue

In the following, we present some results obtained by monitor runs investigating ANSAware performance. Figure 4.1 shows average response time of a sequence of RPCs being sent to the same server depending on the size of the client REX buffer.

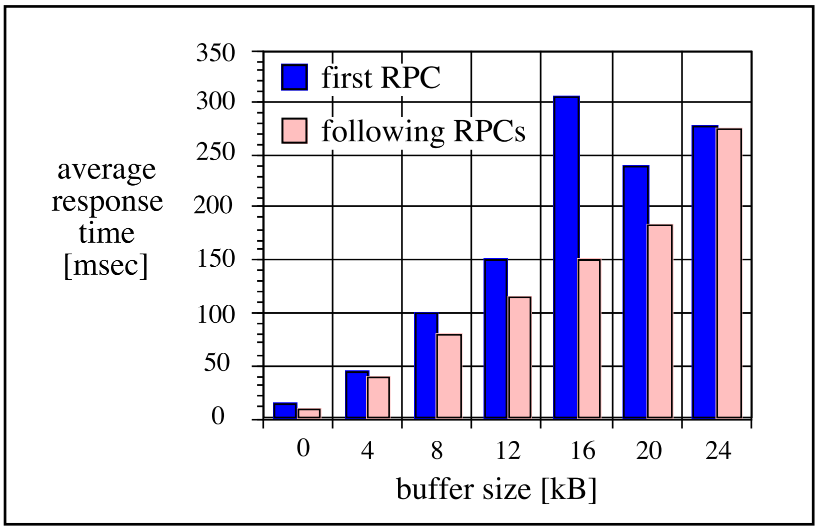

Figure 4.1. Average response time depending on REX buffer size

It can be recognized, that the first RPC for each cooperation lasts longer due to session and channel management within ANSAware. Values for the average response time grow almost constantly with buffer size. It is assumed for all RPCs that call processing at the server takes no time and only one client is attached to a server. Resulting values are comparable to those obtained for DCE RPC in [RaSc93].

4.2 Monitoring import requests to the ANSAware trader

The following three experiments measure the mean response time of import requests to the ANSAware trader. Figure 4.2 shows portions of the average response time of significant subtasks while processing a trader import request.

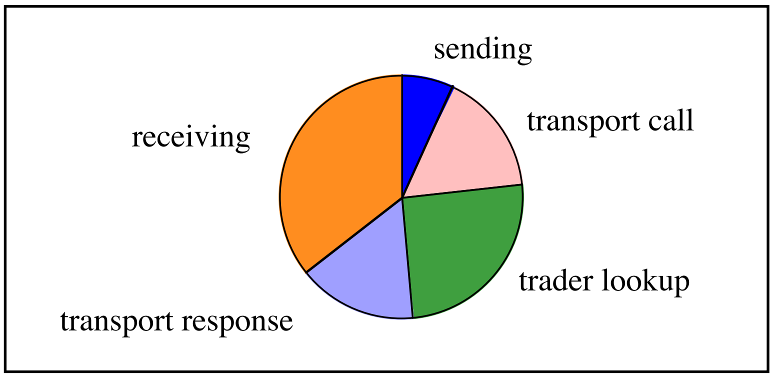

Figure 4.2: Portions of time for a trader import

For a small number of service offers (here for 9) only a fourth part is needed for looking up the trader service database. The time for database lookup will be certainly larger for bigger service databases, as we will show in the next experiment. Even more interesting is the fact, that time required receiving a call until it is performed is longer than the time required for transport service usage.

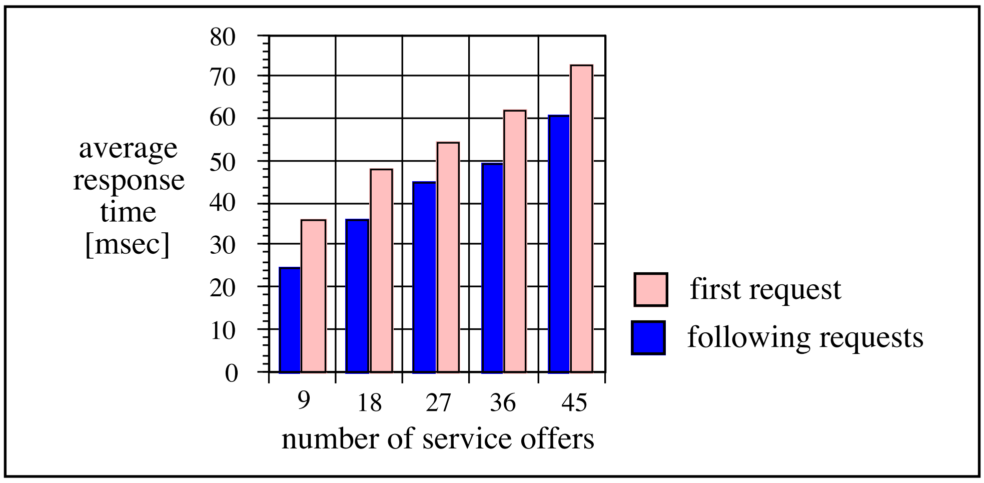

Figure 4.3. Average response time depending on service database size

Further investigation have shown, that REX processing and thread scheduling are responsible for this delay. The major time is used for REX processing, especially when fragementation of the REX buffer is required. We have investigated the influence of service database size in more detail. Figure 4.3 shows the average response time of a trader import request depending of the number of service offers in the service database. It can be concluded from the results obtained, that response time linearly increases with the size of the trader database. This phenomen is not very surprising, because the ANSAware trader always checks all service offers for matching. In addition, a constant amount of time has to be added due to administration purposes. In order to investigate an dependency between the average response time of a trader import request and the number of clients attached to a trader, we did several experiments. The results are shown in figure 4.4. For more than three objects the average response time increases strongly until the converges for 9 and 10 clients.

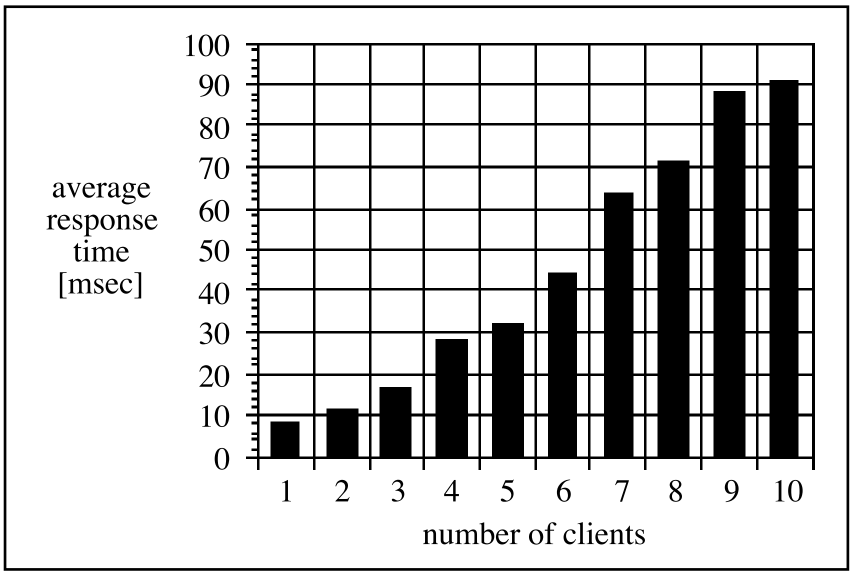

Figure 4.4. Average response time depending on number of clients

5. CONCLUSIONS

Available distributed computing platforms offer a rich functionality to provide distribution transparency to application programmers. In return programmers abandom some control over their application performance. In order to find performance bottlenecks, we developed a distributed monitor for the ANSAware platform. It has been designed to be used for general monitoring tasks, but it also supports special features for performance monitoring. Because analysis is carried out post-mortem, global times can be estimated. Therefore we improved existing methods.

Using ANSAmon we investigated several aspects of the ANSAware platform inter-connect via LAN. At first we measured performance of the REX protocol for inter-process cooperation. The resulting response time values are comparable to those measured for DCE

RPC. In addition, we checked the performance of import requests processed by the ANSAware trader. For traders managing a large number of service offers, looking them up takes most of the response time. In order to achieve more efficient service offer database lookup, service offers should be clustered reasonably, e.g. with respect to service types. In addition, clustering service offers into contexts allow a finer granularity for trader request processing than it is implemented right now. Another performance bottleneck is the REX protocol execution. Usually REX parameter values are fixed. For making efficient use of REX, parameters need to be adapted at run-time according to application requirements.

Until now we only investigated trader cooperation via LAN, but we are preparing monitoring trader cooperation over WAN. Since ANSAmon only offers post-mortem analysis at present, it will be investigated, which performance metrics might be obtained at run-time in order to achieve reasonable on-line analysis. With on-line analysis it would be possible to to efficiently manage applications and services by automatic managers, see [MePo94], [PKM94], [PK94].

REFERENCES

[ANSA] ANSAware Release 4.1, Manual Set, Document RM.102.02, February 1993

[BiNe84] Birell, A.; Nelson, B.: Implementing Remote Procedure Calls. ACM Transactions on Computer Systems, Vol. 2, No. 2, Februar 1984, pp. 39-59

[ChSu92] Chiu, D.; Sudama, R.: Network Monitoring Explained - Design and Application. Ellis Horwood 1992

[DCE] Open Software Foundation: Introduction to OSF DCE. Prentice Hall 1992

[DHH+87] Duda, A.; Harrus, G.; Haddad, Y. et al: Estimating Global Time in Distributed Systems. Proceedings of 7th International Conference on Distributed Computing Systems, Berlin 1987, pp. 299-306

[HKL+90] Hofmann, R.; Klar, R.; Luttenberger, N. et al: Integrating monitoring and modelling of performance evaluation methodology. In: Haerder, T.; Wedekind, H.; Zimmermann, G. (eds.): Design and Operation of Distributed Systems, Springer 1990, pp. 122-149

[Hei94] Heineken, M.: Performance Analysis of the Communication Components of the ODP Prototype ANSAware (in german). Diploma Thesis at Aachen University of Technology, Nobember 1994

[ICT93] Ibe, O.; Choi, H.; Trivedi, K.: Performance Evaluation of Client-Server Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 11, November 1993, pp. 1217-1229

[JLS+87] Joyce, J.; Lomov, G.; Slind, K. et al: Monitoring Distributed Systems. ACM Transactions on Computer Systems, Vol. 5, No. 2, May 1987, pp. 121-150

[Jai91] Jain, R.: The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurement, Simulation and Modelling. Wiley 1991

[Kle75] Kleinrock, L.: Queueing Systems - Volume 1: Theory. Wiley 1975

[Kob78] Kobayashi, H.: Modeling and Analysis: An Introduction to System Performance Evaluation Methodoloy. Addison Wesley 1978

[Lam78] Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, Vol. 21, No. 7, July 1978, pp. 558-565

[LKG92] Lange, F.; Kroeger, R.; Gergeleit, M.: JEWEL: Design and Implementation of a Distributed Measurement System. IEEE Transactions on Parallel and Distrbuted Systems, Vol. 3, No. 6, November 1992, pp. 657-672

[MePo93] Meyer, B.; Popien, C.: Concepts for Modeling and Evaluation of ODP architectures (in german). In: Walke, B.; Spaniol, O. (eds.): Measurement, Modeling and Evaluation of Computing and Communication Systems (MMB´93), Aachen 1993, Springer 1993, pp. 77-89

[MePo94] Meyer, B.; Popien, C.: Defining Policies for Performance Management in Open Distributed Systems. Proceedings of 5th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM´94), Toulouse 1994

[Mil89] Miller, B.: Performance Characterization of Distributed Programs. Ph. D. Dissertation, University of California, Berkeley, 1984

[OMG] Object Management Group: The Common Object Request Broker Architecture: Architecture and Specification. Revision 1.2, December 1993

[ODP p2] ISO/IEC JTC1/SC21 N7988 Information Technology - Open Distributed Processing - Basic Reference Model - Part 2: Descriptive Model. Committe Draft, December 1993

[ODP p3] ISO/IEC JTC1/SC21 N8125 Information Technology - Open Distributed Processing - Basic Reference Model - Part 3: Prescriptive Model. Committe Draft, December 1993

[OSI Man] ISO/IEC IS 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework. International Standard, November 1989

[PK94] Popien, C.; Kuepper, A.: A Concept for an ODP Service Management. Proceedings of IEEE/IFIP Network Operations and Management Symposium, (NOMS´94), Kissimmee/Florida 1994

[PKM94] Popien, C.; Kuepper, A.; Meyer, B.: Service Management - The Answer of new ODP Requirements. In: Meer, J. de; Mahr, B.; Storp, S.: Open Distributed Processing II (ICODP´93), Berlin 1993, North Holland 1994, pp. 408-410

[PMS94] Popien, C.; Meyer, B.; Sassenscheidt, F.: Efficient Modeling of ODP Trader Federations using P^2AM. (in german). In: Wolfinger, B. (ed.): Innovations for Computer and Communication Systems. Springer 1994, pp. 211-218

[RaSc93] Rabenseifner, R.; Schuch, A.: Comparison of DCE RPC, DFN-RPC, ONC and PVM. In: Schill, A. (ed.): DCE - The OSF Distributed Computing Environment, Karlsruhe 1993, LNCS 731, Springer 1993, pp. 39-46

[Rol94] Rolia, J.: Distributed Application Performance, Metrics and Management. In: Meer, J. de; Mahr, B.; Storp, S.: Open Distributed Processing II (ICODP´93), Berlin 1993, North Holland 1994, pp. 235-246