

HPCS : Client-Server Support for High Performance Computing

Inauguraldissertation

zur

Erlangung der Würde eines Doktors der Philosophie

vorgelegt der

Philosophisch-Naturwissenschaftlichen Fakultät

der Universität Basel

von

Gérald Jean-Pierre Prétôt-Eckenstein

aus Le Noirmont, JU

Basel, im Februar 1999

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag der Herren

Prof. Dr. Helmar Burkhart

Prof. Dr. Béat Hirsbrunner

Basel, im April 1999

Prof. Dr. S. Schmid, Dekan

Research Reports in Computer Science

Band 5

Gérald Jean-Pierre Prétôt

**HPCS: Client-Server Support
for High-Performance Computing**

Shaker Verlag
Aachen 1999

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Prétôt, Gérald Jean-Pierre:

HPCS: Client-Server Support for High-Performance Computing/

Gérald Jean-Pierre Prétôt. - Als Ms. gedr. - Aachen : Shaker, 1999

(Research Reports in Computer Science ; Bd. 5)

Zugl.: Basel, Univ., Diss., 1999

ISBN 3-8265-6700-5

Copyright Shaker Verlag 1999

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 3-8265-6700-5

ISSN 1436-6967

Shaker Verlag GmbH • P.O. Box 1290 • D-52013 Aachen

Telefon: 0049/2407/9596 - 0 • Telefax: 0049/2407/9596 - 9

Internet: www.shaker.de • eMail: info@shaker.de

Abstract

Distributed and parallel computing are well-established and successful fields. In both certain levels of standardization have been achieved and reliable tools are available for software development and deployment. However, so far hardly any effort has been made, to *integrate* the two worlds in a generic way. This is the goal of the work presented here.

This thesis introduces a new model for *client-server support for high-performance computing* called *HPCS*. The model offers a generic approach to making parallel, high-performance implementations of *compute services* accessible to sequential clients in the form of software *components*. It exploits the abstraction provided by *parallel data structures* and *parallel procedures* to allow *transparent* access from remote clients. In HPCS parallel data structures are presented as *abstract data objects* and parallel procedures as *services*. Access is provided using standardized client-server *interfaces* such as the ones specified by the *CORBA* standard. This is done in a way that does not require the service programmer to be concerned with the client-server interface nor the client programmer with the parallel nature of the service implementation.

Call-persistent data objects and a minimal set of methods for their manipulation are proposed as a mechanism to work with parallel data structures that remain on the server between different service calls. A *parallel request handler* split into two components, the *Multiplexer* and the *Dispatcher*, provides transparent access to parallel services.

ACS, a concrete implementation of HPCS targeting the parallel programming language *ALWAN* and *CORBA* implementations is presented. Thus, ACS enables writing high-performance services in ALWAN which are translated into executable parallel programs and interfaces that allow access to the services from CORBA clients.

Sources of *overhead* are discussed and some measurements are presented that provide a feeling for the possible behavior of an HPCS application. Examples from the fields of *volume rendering*, *signal processing* and *computational fluid dynamics* demonstrate how ACS is used in applications.

Preface/Roadmap

The presentation of our work is divided into five parts:

Introduction The general ideas and motivation for our approach to client-server support for high-performance computing, are discussed. We have a brief look at the two technologies, the integration of which is the goal of this work: distributed and parallel computing, *CORBA* and *ALWAN* in particular. Part I concludes with an overview of work that is related to ours.

Design and Implementation We introduce an abstract approach to the integration of high-performance computing and client-server infrastructure (HPCS), and present *ACS*, an implementation for the *ALWAN* parallel computing system and *CORBA*. The *ACS* server and client programmers' views are described. Finally, we discuss the overhead incurred by HPCS and some rough measurements made in the *ACS* environment to give the reader a general feeling for the possible behavior of HPCS applications.

Demonstration Applications Three applications using parallel servers are presented: our main example, a 3D volume rendering application from the medical domain (*RDVOL*), but also a signal processing library/application (*LIBSIG/IDAHO*), and a simple, educational computational fluid dynamics code (*CFD*). These applications were used as test cases for *ACS* and to demonstrate how it works.

Future Work and Conclusions We attempt a look at how this work could be extended, conclude from the presented work the position and value of our approach and summarize our experience during the development of *ACS*.

Appendix The appendix contains the ALWAN-to-IDL mapping and the source code (IDL) of the demonstration applications, as well as source code of the program used for measurements. Raw data from the timing measurements is also listed.

Acknowledgement

I feel grateful to a large number of people who have accompanied and supported me during my research and while writing this thesis.

First of all I would like to thank my supervisor (and more) Prof. Dr. Helmar Burkhart for giving me the opportunity to work in this exciting field, for providing guidance when needed but also the environment and freedom to pursue my own ideas, which made the work a truly enjoyable experience. I would also like to thank Prof. Dr. B eat Hirsbrunner for refereeing the thesis and for his valuable and much appreciated feedback.

A very special thank goes to my family, to my wife, Sandra, and my son, S ebastien, who both had to suffer through some difficult times, especially towards the end of my work. Although I must have often been quite unbearable, they still kept supporting me as only a loving family can.

My thanks also go to my parents Leny and Fran ois Pr et ot and my brother Ren e, my dear parents in-law, Monique and Heinz Eckenstein and my brothers in-law, Sacha and Fabian. They provided the net that sometimes kept me from falling.

Doing this work would not have been half the fun without the valuable discussions with my colleagues at the Computer Science Department, my family's dear friend and author of ALWAN, Dr. Guido H achler, Dr. Edgar Lederer, the discussions with whom I will miss as much as the famous chocolate cakes, Robert Frank, who always seemed to be there and know an answer even for the stupidest questions and who kept the computers and networks running (most of the time, anyway) and Dr. Birgit Westermann, who is the author of the parallel Rdrv code and had to endure many long discussions about ALWAN, ACS and the amount of disk space available on the file server.

Dr. Gonzalo Travieso maintained the ALWAN tools and removed a number of idiosyncrasies in the code (aka *bugs*) without which work on ACS would have been impossible. Kay Ruchti wrote the core of the parallel CFD code and was always willing to discuss and give feedback.

Other important sources for feedback were Dr. Niandong Fang, Dr. Beat H ormann, Dr. Robert Sinclair and Dr. Walter Kuhn. I am also grateful for Romeo Dumitrescu's technical support that was always available when needed.

Last but by no means least of the people at the Computer Science Department whom I owe, are our former secretary, Regula Kneub uhler and our current secretary Karin Liesenfeld. The good services and sup-

port that they provided are innumerable. I really don't know how they put up with us sometimes and still managed to bring color and sunshine into our offices.

I would like to thank the Swiss Science Foundation and the Swiss Federal Office of Education and Science for sponsoring parts of this work¹ and some of our collaborators in the projects, especially Patrick Jacques, Jean-Luc Viellé and Chi Ngo Duc of Spacebel Informatique S.A., Déphine Délavaux of Aérospatiale and Pierrick Beaugendre of the IRISA.

Grzegorz Mysliwiec worked on the parallel code for the LIBSIG server.

There are a number of people at Hoffmann-La Roche where I continued to work part time during the full duration of my work on the thesis. Dr. U. Goetz who very much supported my first steps "going back to school".

Prof. Dr. Don Kaiser, Mark Peeters, Dr. Alfred Steinhardt, Nathalie Legrenzi and Claudia Bohnet, my managers, who were kind enough to tolerate my part time presence in the office and who provided a work environment that allowed me to wear more than two hats without going absolutely crazy.

Much of what is left of my sanity I owe to my dear friends Maja, Tobi and Patrick of the *MMM*, with whom I so often enjoyed the pleasures of music, from which I drew pure energy.

There are so many people whom I have the honor of calling my friends, who accompanied me through this time and who continue to be an important part of my life. Any list would be incomplete. Thank you all!

¹Projects: SPP IF 5003-034357 and ESPRIT HPCN project 21029, *PACHA*

Contents

I	Introduction	3
1	Integrating CS and HPC Computing	5
1.1	Client-Server Support for High Performance Computing . . .	5
1.2	Motivation	6
1.3	How does our approach differ from others?	9
2	Building Blocks	11
2.1	Client-Server Computing	11
2.1.1	What it is about	11
2.1.2	Tasks of Client-Server Computing Infrastructure . . .	13
2.1.3	RPC and OODP	15
2.1.4	CORBA	20
2.2	High-Performance Computing (HPC)	23
2.2.1	ALWAN - The HPC Approach Used in our Work . . .	25
3	Related Work	31
3.1	Legion	33
3.2	Mentat	35
3.3	Globus	36
3.4	Condor	37
3.5	NetSolve	39
3.6	Ninf	40
3.7	RCS	42
3.8	Sciddle	43
3.9	Nexus	44
3.10	HPC++	44
3.11	POOMA	45
3.12	PARDIS	46

3.13 PAWS	47
3.14 Cobra	48
3.15 Others	49

II Design and Implementation 51

4 Design and Implementation of an HPCS Environment 53

4.1 Architecture	54
4.1.1 Multiplexer and Dispatcher	54
4.1.2 Call-Persistent Data	56
4.2 The ACS Implementation	57
4.2.1 Source Code Generation	58
4.2.2 The CORBA Interface	58
4.2.3 The Multiplexer implementation	62
4.2.4 The Dispatcher implementation	62
4.2.5 The call-persistent data implementation	64
4.2.6 Tools	66
4.3 Platforms on which ACS is currently implemented	67

5 ACS Programmer's View 69

5.1 ACS Server Programmer's View	69
5.1.1 The ALWAN Main Module	69
5.1.2 The Module Body	70
5.1.3 Parallel Services	70
5.1.4 TOPOLOGY parameter list	71
5.1.5 Global Variables	71
5.1.6 Type Templates for Partitioned Types	72
5.1.7 Input/Output	72
5.1.8 Compiling for ACS	73
5.1.9 Examples	74
5.2 ACS Client Programmer's View	76
5.2.1 ACS Servers	76
5.2.2 ACS's Call-Persistent Data Objects	76
5.2.3 ACS High Performance Services	78
5.2.4 Programming with ACS	79
5.2.5 Examples	79

6	HPCS Overhead	83
6.1	Suitability Considerations	83
6.2	HPCS vs Standalone Implementation	86
6.3	Measurements	87
6.4	How the measurements were performed	88
6.5	Results	90
6.5.1	CORBA and Network	90
6.5.2	ACS-internal overhead	91
6.5.3	Influence of Call-Persistent Data on Performance	95
III	Demonstration Applications	99
7	Sample Applications	101
7.1	RDVOL	101
7.1.1	Application Architecture	102
7.1.2	RDVOL Execution Structure	104
7.1.3	Timings	105
7.1.4	Adaption Effort	108
7.2	IDAHO/LIBSIG	109
7.2.1	IDAHO Application Structure	110
7.2.2	Execution Structure	110
7.3	CFD	113
7.3.1	Execution Structure	114
7.3.2	Timings	114
IV	Future Work and Conclusions	117
8	Future Work	119
8.1	Possible Improvements and Extensions to ACS	119
8.1.1	Type Mapping	119
8.1.2	Broadcasts and Distribution	121
8.1.3	Bulk Data Transfers	121
8.1.4	Exception Handling	123
8.1.5	Alternatives to Call-Persistent Data	123
8.1.6	Additional Examples	123
8.1.7	Beyond ACS	124
8.2	HPCS in ‘Standard’ Parallel Programming Environments	124

9	Conclusion	127
9.1	Overall Conclusions	127
9.2	The ACS Experience	128
V	Appendix	131
A	ALWAN to IDL Mapping	133
B	Measurement Source Code	137
B.1	ALWAN Code: acsTimer.asf	137
B.2	IDL Code: acsTimer.idl	139
C	Demonstration Application IDL Sources	143
C.1	rdvol.idl	143
C.2	Libsig.idl	148
C.3	cfid.idl	151
D	Raw Timing Data	153
E	Product List	157
	Bibliography	159
	Index	166

List of Figures

1.1	HPCS	7
2.1	Schema of an RPC call	16
2.2	Structure of ORB interfaces	22
2.3	ALWAN <i>Torus</i> TOPOLOGY Communication	27
2.4	ALWAN Data Partitioning	28
2.5	ALWAN Local vs. Global View	29
4.1	Overview of HPCS Architecture	54
4.2	Communication architecture of the main ACS components. 56	
4.3	ALWAN-to-IDL Example	59
4.4	ACS Code Generation	67
6.1	ALWAN INPUT vs ACS Put	93
6.2	ACS Put with 2 and 6 processors	94
6.3	Put: Client vs Server Times	96
6.4	Influence of call-persistent data	98
7.1	Architecture of the parallel volume rendering application.	103
7.2	The RDVOL Java client - displayed in the Netscape browser	106
7.3	A screen shot of the IDAHO Java client	111
7.4	The scfd Java client - displayed in the Netscape browser .	113
8.1	ACS Languages and Type Translations	120

List of Tables

2.1	IDL-to-Java Example	21
4.1	ALWAN components and their IDL translations	60
5.1	files generated by: <code>tiana test.asf -cs</code>	73
6.1	ACSvs standalone ALWAN	86
6.2	Services used for in timing measurements	89
7.1	Schema of a typical RDVOL run	105
7.2	Overview of RDVOL Service Arguments	107
7.3	RDVOL Times	108
7.4	Execution Structure of an IDAHO	112
7.5	IDAHO service paramters	112
7.6	Schema of a typical CFD run	114
7.7	Parameters of CFD Services	115
7.8	Measurements for CFD	115
A.1	Simple Types	134
A.2	Aggregate Types	134
D.1	acsTimer: C++ client times	154
D.2	acsTimer: Java client times	154
D.3	acsTimer: OrbixWeb Java client times	155
D.4	acsTimer: ACS times, Sun	155
D.5	acsTimer: ACS times for Put, Sun	156
D.6	acsTimer: ACS times, GigaBooster	156
D.7	acsTimer: ACS times for Put, Gigabooster	156