submitted to PDPTA 2005, presenting author: Christian Rehn

Constituting the Need for Flexibility in Distributed Operating Systems

Markus Pizka
Institut für Informatik - I4
Technische Universität München
Boltzmannstr. 3
Germany - 85748 Garching
pizka@in.tum.de

Christian Rehn
Institut für Informatik - I13
Technische Universität München
Boltzmannstr. 3
Germany - 85748 Garching
Phone/Fax: +49 (89) 289 {18580/18557}
rehn@in.tum.de

Abstract *The intrinsic goal of any operating system (OS) is to free the application level from tasks that are either repeating or hard to accomplish. In the 80s and 90s there have been strong efforts to develop distributed OS providing fully location transparency. Although these projects delivered valuable results on certain aspects of distributed computing, satisfactory general purpose distributed resource management at the OS level could not be achieved. We argue that the performance gap between local and remote operation is the major obstacle for truly transparent distributed resource management. This performance gap can only be bridged with significantly increased flexibility at the OS level. At the same time constant overhead must be avoided.*

Keywords: distributed systems, operating systems, high performance

1 Introduction

Achieving peak performance in parallel systems is challenging yet even more difficult in the case of MIMD [1] like distributed systems [2], consisting of networks of workstations with no direct access to remote memory. Trying to build a general purpose distributed operating system (OS) that delivers at least reasonable performance seems to push these well-known difficulties to the extreme.

The purpose of an OS in general is to release the application level from difficult, repeating, or – due to rights – impossible tasks which can be provided without significant loss of quality or performance as a service to the application level. In this sense, a distributed OS should provide fully transparent management of all distributed resources, such as processors, memory and communication bandwidth. The application level should neither be concerned with the distribution of load nor should it have to deal with the placement of data objects within the network. It should solely specify an explicitly parallel computation and let the OS arrange the actual distribution. Notice, in contrast to distributed runtime systems full transparency is crucial for distributed OS. The reason for this is that the interplay between application and OS level decisions would not be controllable if there was only partial transparency.

Of course, the idea of fully transparent distributed resource management at the OS level is all but new. From the late 80s through the 90s a huge amount of work has been invested into the design and implementation of distributed OS with systems like Amoeba, Muse, Apertos, Mungi, Sprite, Chorus, MoDiS, and Spring [3, 4, 5, 6, 7, 8, 9, 10]. Without any doubts all of these projects contributed significantly to the advancement of the field. But as we take a look back on these initiatives 10 years later we have to admit that OS have failed to live up to their expectations. Research interest is declining and there are still hardly any commercial distributed OS products despite of the increasing demand.

Outline

In this paper, we try to give an answer to the questions why there is no such OS available, yet and whether such an OS was feasible at all.

Section 2 argues that the performance gap between local and remote operation is the major source of problems and discusses its impact on distributed OS. Section 3 proposes significantly increased flexibility at the OS level as a mean to cope with the performance gap. Finally section 4 summarizes this paper.

2 Distributed OS and Performance

Obviously there are several technical as well as non-technical reasons for the absence of distributed OS, such as market pressure and the need for compatibility.

But even in the presence of increasing popularity of interpreted languages like Java [11], *performance problems* still remain a major obstacle for further progress in the distributed OS field. This is because suboptimal resource management decisions in a distributed OS may not just cause some performance losses but orders of magnitudes.

2.1 The 10^5 Gap

The root of this phenomenon is the enormous gap between local (i. e. direct memory or even cache) and remote (i. e. network) access. Table 1 compares the latency and bandwidth of local versus remote accesses on an Intel Pentium 4, 3.4 GHz and 100 MBit/s Fast-Ethernet resp. 1 GBit/s Gigabit Ethernet. Particularly for small messages, the speed of local versus remote access may differ by a factor of up to 10^5.

In practice, this could mean that a parallel computation runs up to 1 day and 3 hours instead of only 1 second, if an object is distributed inappropriately! Of course, real situations are usually less dramatic but well-known effects, such as *false sharing* [12], demonstrate similarly drastic impacts of inadequate resource management decisions. Clearly, an OS must avoid the 10^5 performance penalty as far as possible by keeping remote communication to a minimum yet trying to maximize the utilization of distributed resources.

First, a decision to distribute must be based on estimates that indicate that the benefits of distribution outweigh its drawbacks. This is only possible if the OS possesses additional information about the structure of the computation. Second, the OS must carefully distinguish between local and remote operation and treat them with sets of completely different strategies. This is because there is no single suitable management strategy in distributed environments. In contrast to non-distributed systems, minor deviations from the optimal strategy may easily cause unacceptable performance degradations due to the 10^5 performance gap.

In [13] we demonstrated how accesses to remote objects can be significantly optimized at the OS level by dynamically switching between object replication, object migration, and remote invocation. The criteria for the dynamic selection of one of these mechanisms are the frequency of reads and writes. Obviously this is just one important example for the need of flexibility. The results can easily be transfered to other tasks, such as partitioning in DSM systems [14], adjusting the granularity of transport units [15] and thread creation.

Thus, the 10^5 gap must be met with a great deal of flexibility by the OS. Uniform resource management strategies will always deliver unacceptable performance in some situation.

2.2 The Impact of Overhead

Now, the need for flexibility is widely accepted and the combination and integration of different strategies is intensively studied in the literature, see for example [16, 17, 18, 19]. But

latency (nanosec.)				**bandwidth** (MB/s)			
L1	mem	TCP (Fast)	TCP (Gigabit)	L1 (read)	mem	TCP (Fast)	TCP (Gigabit)
4	56.1	$2.5 * 10^5$	$2.4*10^5$	47787	4322	10.69	51.32

Table 1: Performance gap

unfortunately, flexibility itself causes new troubles for OS.

With runtime systems it is up to the application level to pick the right strategy at the right time. The programmer is expected to know about the structure of the computation. He picks adequate strategies and hard-wires these decisions into the application. Thus, there is usually no costly decision making during runtime. In contrast to this, an OS with fully transparent distributed resource management has to make and enforce decisions during runtime [20]. For this, it has to collect and evaluate information, it has to switch between strategies, and it must make use of indirections to enable dynamic switching. All of this causes overhead!

Besides the lack of flexibility, constant overhead is the second major source of performance problems in distributed OS. Comparisons of distributed systems supporting location transparency with shared memory multiprocessor systems expose this difficulty. For example, Levelt [21] used parallel matrix multiplication to compare the performance of a multiprocessor system with the distributed system Orca [22] using identical hardware. Although the distributed system delivered perfect relative speed-ups, one can also see from table 2 that 5 nodes (169.9) are needed just to achieve the performance of a similarly powerful uniprocessor (180.5) resp. the shared mem multiprocessor with one node only.

Many other distributed OS suffered from the same problem which is concentrating too much on relative speed-ups instead of absolute ones or at least acceptable absolute slowdown. With distribution and relative speedups in mind the system is extended with additional techniques, such as DSM, RPC, distributed thread creation, dynamic load balancing, etc. This concepts are integrated in a straight-forward manner using indirections, argument marshaling, and alike. But all of this causes additional overhead resulting in a slowdown instead of the envisaged speed-up. It is clearly trivial to achieve relative speed-ups if local execution is just slow enough.

It should be evident, that hardly anyone is willing to use a distributed OS that requires multiples of resources just to achieve the performance one would get without distribution. Even if we are not focused on performance but want to use distribution for example for increased reliability, a n times slow-down caused by the OS will never be accepted. The user would still rather waive OS support and use an application level controlled runtime system.

2.2.1 Amdahl's Law Revised

Now, one could argue that an absolute slowdown of 5 is indeed acceptable because the investment will pay-off if a large numbers of nodes (64, 128, ...) is used.

If we keep in mind that realistic parallel algorithms contain a significant amount of sequential work and combine this with Amdahl's law [23]:

$$S_r = \frac{n}{1 + (n-1) * t_s}$$

which gives an upper bound for the relative speedup S_r dependent on the fraction of sequential work t_s and the number of nodes n, then it becomes evident that it is impossible to deliver satisfactory distributed performance if there is significant constant overhead regardless of the number of nodes used.

To study the impact of additional overhead with a simple analytical model, we extend Amdahl's law with constant local and synchronous remote overhead. Constant local overhead is caused by a wide range of changes to the OS to prepare for distribution. This includes in-

matrix multiplication (seconds)					
nodes	1	2	3	4	5
Orca multicast	810.3	410.6	279.1	209.8	<u>169.9</u>
shared mem. multiproc.	<u>180.5</u>	90.5	60.8	45.9	36.6

Table 2: Comparison of matrix multiplication

direct calls, handling data on heap instead of stack or registers to allow for distribution, changed inter process communication (e. g. indirect communication via coordinators instead of signals) or using messages instead of shared memory. Synchronous remote overhead is everything needed to actually perform remote operations as they appear in the computation. One example is the effort needed to accomplish the remote creation of a thread instead of its local creation.

Let t_s be the sequential and t_p the parallel amount of work, $t_s + t_p = 1$. Let n be the number of processors, l the constant local overhead and y the additional costs for remote operations. S_a approximates the absolute speedup by augmenting Amdahl's law with l and y.

$$S_a(l, y) = \frac{t_s + t_p}{t_n} = \frac{n}{(t_s * (n - y) + y) * l}$$

As one can see, minimizing the impact of constant overhead is crucial for achieving reasonable performance of the system.

Figure 1 shows S_a for $t_s = 10\%$ and different combinations of l and y. l, the local overhead, has a dominant impact on the performance of the system. If l is only 4, then 32 nodes are needed to achieve an absolute speed up of 2 already. No further significant improvements are possible by using more than 32 nodes.

This indicates that the evaluation of management strategies based on measurements of relative speed ups on a few nodes must be taken with care. The extrapolation to larger numbers of nodes is misleading. Significant slow-downs on one processor can usually not be compensated, even if one is willing to employ large configurations.

Note the fact that only the impact of y can be decreased with faster network connections. The dominant factor l is independent of hardware properties. This means that better and faster networks are no solution to the distributed OS performance problem.

2.2.2 Gustafson-Barsis Revised

Amdahl's law is based on the assumption that the size of the problem remains constant independent of the number of nodes. This does not adequately reflect situations, where the increased computational power is used to increase the size of the problem (e. g. multiply larger matrices). Gustafson reevaluated Amdahl's law [24] to allow scaling the problem size with the result $S_{r_g} = 1 + (1 - n) * t_s$. Again, S_{r_g} does only approximate the relative speed up, not the absolute effect. For this we have to reconsider l and y leading to following estimate:

$$S_{a_g} = \frac{t_s + n * t_p}{(t_s + y * t_p) * l} = \frac{t_s * (1 - n) + n}{(t_s * (1 - y) + y) * l}$$

Figure 2 shows S_{a_g} for $t_s = 10\%$ and different values of l and y. On the one hand it can be seen that if increased computing power is used to compute larger problems then local overhead does not a priori restrict actual performance gains. On the other hand, the performance degradation due to local overhead remains proportional independent of problem size and number of nodes. Thus, even if the problem size scales perfectly with the size of the configuration, which is rather unrealistic in a general purpose OS, local overhead still has to be avoided as far as possible.

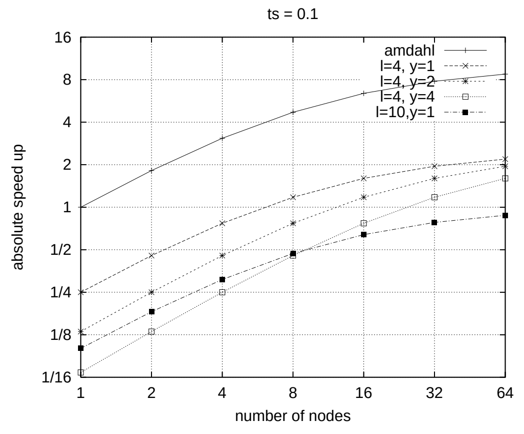

Figure 1: Amdahl's law and magement overhead

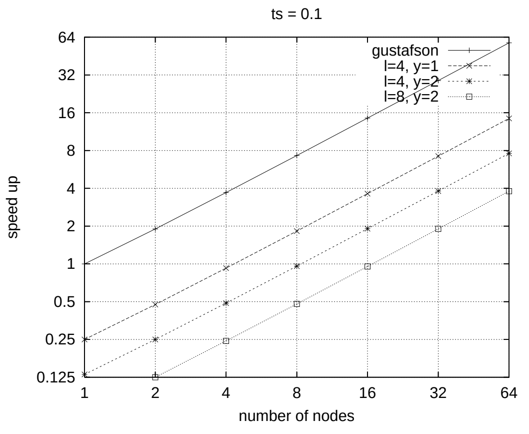

Figure 2: Gustafson-Barsis and const. overhead

3 Operating System Design

According to the argumentation above, distributed OS are in a serious dilemma. On the one hand, uniform management strategies are unsuitable because of the 10^5 performance gap. Any single strategy will deliver unacceptable performance in some real situation. On the other hand, flexibility is expensive. It causes overhead that puts distribution in question.

3.1 Are Distribute OS Worth Considering?

It is legitimate to ask whether distributed OS with fully transparent resource management are realistic, anyway. Maybe distribution management should simply be left to the application level.

One must keep in mind that programmers may eventually make mistakes or do not even have enough information about the execution environment when writing the distributed program. This is frequently the case when the configuration changes after the distributed application was written. In this situation the assumptions and decisions concerning physical distribution are violated causing unpredictable performance effects. But since these decisions are coded into the application they must be changed manually, which is costly, time-consuming, error-prone, and should therefore better be left to the OS level.

For increased quality (portability, reliable, number of bugs, . . .) and reduced costs of parallel and distributed applications from a Software Engineering perspective it is worthwhile to further investigate possibilities to design distribute OS with the desired degree of transparency and reasonable performance although setting this goal seems ambitious. Here, it is important to stress "'reasonable"'. From the considerations above, it should be clear that distributed OS can hardly be the means to achieve peak performance. But we state that a general purpose distribute OS with acceptable performance is feasible and useful. To achieve this it is necessary to investigate innovative ways to achieve *flexibility* while preserving *low overhead*.

3.2 OS = Compile-Time and Run-Time

We consider all software tools involved in resource management decisions as the means to implement an OS [25, 26]. This includes the OS kernel, runtime libraries, loader, linker, inlined and runtime generated code, and the compiler. The effectiveness of the whole OS depends on both, the services provided by each tool and their interplay. Projects like TreadMarks [27] demonstrated the performance gains that can be achieved through the integration of these different tools.

This integration is twofold. First, information should be exchanged between all tools involved in resource management as needed. For example, it is somehow incomprehensible that compilers perform sophisticated data flow analyses but the results are thrown away after compilation although the runtime system or OS kernel would need this information to steer the distribution of tasks and data. In turn the OS has to perform runtime monitoring causing avoidable overhead. This is clearly not optimal and must be changed to allow exchange of information between different tools.

The second aspect of integration is the possibility to enforce management decisions with different tools. For example, a parallel entity at the application level could be implemented as a sequential function, a local thread or a remote thread. The decision for one of these possibilities can either be made by the compiler by generating corresponding target code or it can be delayed till link time or further down to a runtime library or even the OS kernel. Flexibility increases with each delay but efficiency is higher the sooner the decision is made. Thus, by integrating the whole tool set into one holistic view of the OS we gain the possibility to balance flexibility with its costs.

4 Conclusion

So far, general purpose distributed and parallel OS have failed to live up to their expectations. In this paper we argued that substantial performance problems are a major reason for this. We identified the 10^5 performance discrepancy between local and remote operation and constant overhead as the major sources of these problems and explained why these problems are particularly difficult to the OS field. We also showed that these problems can not be solved satisfactorily with improved hardware. Instead, great efforts at the OS level are necessary to cope with the 10^5 gap while keeping the additional overhead low.

Based on our analytical performance models in section 2 it is evident that there is still a long way to go before we can expect general purpose distributed OS with reasonable performance. Nevertheless, we think this is a challenging and interesting field of research worthwhile further efforts.

References

[1] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972.

[2] Philip H. Enslow Jr. What is a "distributed" data processing system? *Computer*, 1978(1):13–21, January 1978.

[3] S.J. Mullender, G. van Rossum, A.S. Tanenbaum, R. van Renesse, and H. van Staveren. Amoeba: A distributed operating system for the 1990s. *IEEE Computer Magazine*, pages 44–53, May 1990.

[4] Y. Yokote, A. Mitsuzawa, N. Fujinami, and M. Tokoro. Reflective object management in the Muse operating system. In *Proceedings of the 1991 International Workshop on Object Orientation in Operating Systems*, pages 16 – 23. IEEE Computer Society Press, 1991.

[5] Y. Yokote. The Apertos reflective operating system – the concept and its implementation. In *Proceedings of the Conference on Object-Orientated Programming Systems, Languages and Applications (OOPSLA)*, pages 414 – 434. ACM Press, 1992.

[6] G. Heiser, K. Elphinstone, S. Russell, and J. Vochteloo. Mungi: A distributed single address-space operating system. Technical Report SCS &E Report 9314, School of Computer Science and Engineering, University of New South Wales, November 1993.

[7] J. K. Ousterhout, A. R. Cherenson, F. Douglis, and M. N. Nelson B. B. Nelson. The Sprite network operating system. *Computer*, 21(2):23–36, 1988.

[8] The Chorus Team. Overview of the Chorus distributed operating system. In *USENIX Workshop on Microkernels and other Kernel-Architectures*, Seattle, WA, 1992.

[9] C. Eckert and H.-M. Windisch. A new approach to match operating systems to application needs. In *Int. Conf. on Parallel and Distributed Computing and Systems*, pages 499 – 503, Washington, USA, Oktober 1995.

[10] Graham Hamilton and Panos Kougiouris. The Spring nucleus: A microkernel for objects. In *Proceedings of the USENIX Summer 1993 Technical Conference*, pages 147–160, Berkeley, CA, USA, June 1993. USENIX Association.

[11] Sun Microsystems, Mountain View, CA. *The Java Language Specification*, 1.0 beta edition, October 1995.

[12] W. J. Bolosky and M. L. Scott. False sharing and its effect on shared memory performance. *Proc. of Fourth SEDMS*, Sep. 1993.

[13] H.-M. Windisch. Improving the efficiency of object invocations by dynamic object replication. In H. R. Arabnia, editor, *Proc. of PDPTA*, pages 115 – 131, November 1995.

[14] Kai Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Department of Computer Science, Yale University, New Haven, CT, October 1986.

[15] B. Buck and P. Keleher. Locality and performance of page- and object-based DSMs. In *Proc. of the First Merged Symp. IPPS/SPDP*, pages 687–693, Los Alamitos, March 30–April 3 1998. IEEE CS.

[16] C. Amza, A. L. Cox, S. Dwarkadas, L-J. Jin, K. Raj amani, and W. Zwaenepoel. Adaptive protocols for software distributed shared memory. *Proc. of the IEEE, Special Issue on Distributed Shared Memory*, 87(3):467–475, 1999.

[17] Henri E. Bal and Matthew Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, 6(3):74–84, July/September 1998.

[18] S. Dwarkadas, H. Lu, A. L. Cox, R. Rajamony, and W. Zwaenepoel. Combining compile-time and run-time support for efficient software distributed shared memory. *Proc. of the IEEE, Special Issue on Distributed Shared Memory*, 87(3):476–486, March 1999.

[19] S. Ioannidis and S. Dwarkadas. Compiler and run-time support for adaptive load balancing in software distributed shared memory systems. In *Proc. of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, May 1998.

[20] Christian Rehn. Dynamic Global Scheduling in Cooperative Distributed Systems. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'2003*, pages 1427–1433, Las Vegas, NV, June 2003.

[21] W. G. Levelt, M. F. Kaashoek, H. E. Bal, and A. S. Tanenbaum. A comparison of two paradigms for distributed shared memory. *Software– Practice and Experience*, 22(11):985–1010, November 1992.

[22] H. E. Bal and M. F. Kaashoek. Object distribution in orca using compile-time and run-time techniques. In *OOPSLA'93*, pages 162–177, September 1993.

[23] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS conference proceedings, Spring Joint Computing Conference*, volume 30, pages 483–485, 1967.

[24] J. L. Gustafson. Reevaluating amdahl's law. *Commun. of the ACM*, 31, 5:532–533, 1988.

[25] M. Pizka, C. Eckert, and S. Groh. Evolving software tools for new distributed computing environments. In H. Arabnia, editor, *PDPTA'97*, pages 87–96, Las Vegas, NV, July 1997.

[26] Markus Pizka. *Integriertes Management erweiterbarer verteilter Systeme.* PhD thesis, Technische Universität München, June 1999.

[27] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE COMPUTER*, 29(2):18–28, February 1996.