Replication on multicore considering hierarchy and characteristics of the network

Stefan Kaestle
stefan.kaestle@inf.ethz.ch
Systems Group, ETH Zurich

March 14, 2013

What do you want to enable? Increase algorithm performance on multicores by applying distributed principles, particularly replication. The challenge then is to guarantee consistency. Previous work was using primary copy protocols to do that [5], a passive replication scheme (i.e. nodes do not have to contact all replicas). They do that because they claim that it is not suitable for clients to know about replication details. An atomic broadcast is often used as foundation of such algorithms. It guarantees in reliable delivery of messages to every node.

What problem are you solving, and why is it hard? Three problems:

placement machine dependent

consistency need to consider machine *characteristics*; multicores are *different* from classical distributed systems.

machine characteristics diverse, fast changing, complex. One example is the batch size for sending a broadcast.

What's the related work? People do replication to deal with scalability challenges on multicores, e.g. databases [3, 5] and operating systems [4, 2, 1]. But they don't to it properly. They do not consider machine characteristics or apply the wrong distributed technologies.

What new idea(s) will solve the problems? We do a hierarchical group communication, which implements an atomic broadcast on multicore machines. It has message complexity of $O(\frac{n}{2})$ and time complexity $O(log(n))$ as opposed to algorithms from traditional distributed systems having $O(2)$ message complexity and $O(n)$ time complexity.
Example: sequential consistency on shared memory, do not really need to have primary backup.

How will you go about it? Replicas on NUMA domains, show that overall throughput is higher. Then, use a quorum based approach for consistency. Consistency by doing a hierarchical group communication approach instead of sequential send from same nodes.

How will you know and show it works? Database? Microbenchmark? Compare to primary backup, which is typically used. Measure time required to apply an update.

What is your hypothesis? We can do much better if machine characteristics are considered.

0.1 Mechanisms

0.1.1 Aggregation

Similarly to what has been done in wireless sensor networks (where it also matters to reduce the number of messages, but other reasons: power consumption), we can do aggregation in nodes. In difference to traditional distributed systems, this works, because it is easy to deploy custom software on every node in the network. Classical distributed systems do not typically allow this. Furthermore, reducing the number of messages at the price of higher complexity does not make sense there.

Examples for aggregation: number of nodes agreeing to something, find capabilities (concatenate core ids).

Open questions: How does this work with several senders (everyone is potentially a sender). Maybe something like in navigation systems. Every node has an entry point, which forwards the messages. Nodes are then clustered ...

0.2 Implementation

Build a tree. Every node has parents and children (except for the root and the leafs). The number of children does not really matter for the algorithm. In the init phase, we construct the tree (e.g. broadcast: one root node, all other nodes are children of the root node, multicast: every node has two children, NUMA tree: two layers, two different implementations?

Reducing the number of children also helps in select. It reduces the number of "channels" to poll, and therefore the latency of detecting messages. Instead of

0.3 Experimental results

Current numbers

Tree setup [SK describe]

broadcast algorithm	nos5		gruyere	
	cycles	error	cycles	error
sequential	4096.1	105.2	136751.9	3061.5
batch	2408.0	181.9	57360.8	5271.5

Table 1: Broadcast measurements

Measurements to do next

- Increase number of cores to talk to in every round. Use a separate waitsets, so that we don't have additional overhead from the bindings not used in each concrete measurement.
- Play with batching. Incrementally increase the number of messages send in a batch. I assume this is machine specific, which means, that this is something programmers do not want to deal with. We measure this at dynamically at run-time and find the right batch size.

Minimal cost of sending a tree-broadcast Plot 1 shows the minimal cost of flooding a sub-tree. The group communication is based on a binary tree. Core 0 is the root, cores 1 and 2 are its children etc.

The average cost is really high (so something is wrong with my code). But the minimal numbers show what is possible. The numbers achieved are easily explainable. The cost is increasing logarithmic with the number of nodes reached by the broadcast (as expected). Every level in the tree adds an additional 3500 cycles to the tree. Node 0 takes significantly longer. I don't know yet why that is.

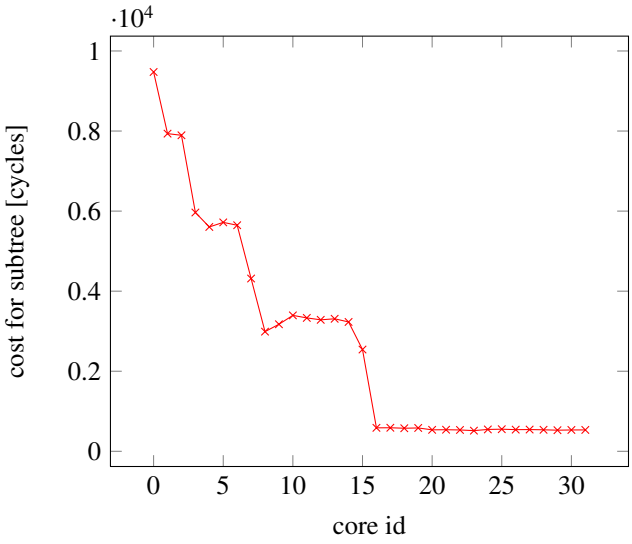
Plot 3 shows some weirdness on gruyere. The cost of the broadcast seems to be growing over time. No clue why.

- Move server to a core != 0
- Use low-level UMP stuff

✓ disable yielding set

- some other item
- run several request in parallel, give them IDs, can use in the measurement struct. Good for everything having transactions (DB, transactional memory, consistency in replication systems)

Figure 1: Plot



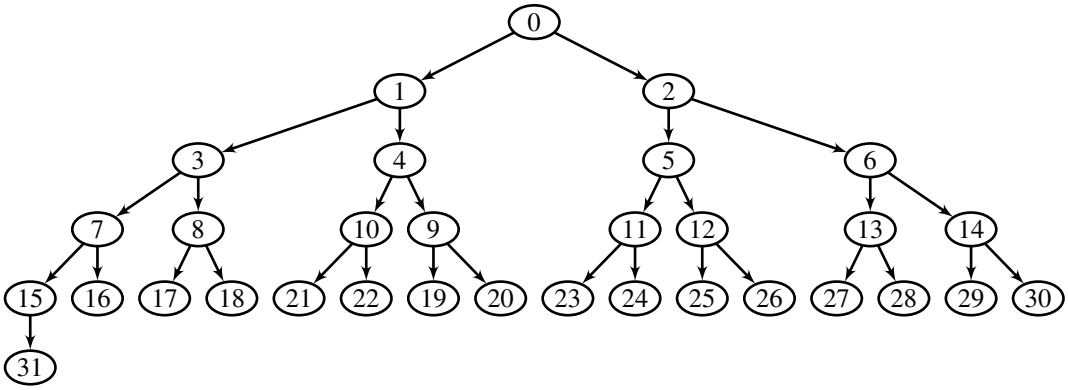
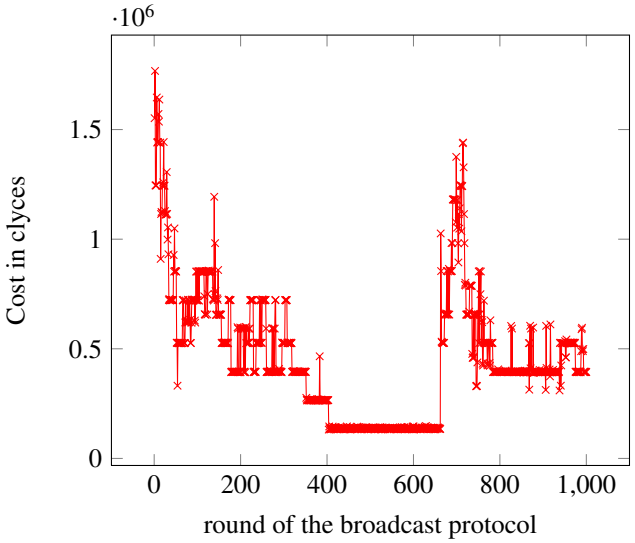


Figure 2: Tree of cores for broadcast on gruyere

Figure 3: Sequential broadcast on gruyere. In every round, we send a request to every other core in the system and wait for an ACK before sending the next message. For the table we only use the results between round 450 and 600, as these seem to be most stable.



References

[1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the 22nd ACM Symposium on Operating System Principles*, pages 29–44, October 2009.

[2] Ben Gamsa, Orran Krieger, Jonathan Appavoo, and Michael Stumm. Tornado: Maximising locality and concurrency in a shared memory multiprocessor operating system. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, pages 87–100, February 1999.

[3] Tudor-Ioan Salomie, Ionut Emanuel Subasu, Jana Giceva, and Gustavo Alonso. Database engines on multicores, why parallelize when you can distribute? *Proceedings of the sixth EuroSys conference*, page 17, 2011.

[4] David Wentzlaff and Anant Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. *SIGOPS Operating Systems Review*, 43(2):76–85, 2009.

[5] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and Gustavo Alonso. Understanding Replication in Databases and Distributed Systems. *Proceedings of the*

20th Internation Conference on Distributed Computing Systems, pages 464–474, 2000.