Brief Announcement: Few Buffers, Many Hot Spots, and No Tree Saturation (with High Probability)

Bradley C. Kuszmaul
MIT CSAIL
bradley@mit.edu

William Kuszmaul
MIT PRIMES
william.kuszmaul@gmail.com

ABSTRACT

In a multistage network, hotspots induce tree saturation. The known solutions employ a variety of techniques, including combining (which works only for certain kinds of messages), feedback damping (which appears to provide low utilization in the absence of hot spots), and large numbers of buffers. In practice, the approach used today is to provide large numbers of buffers: in a P-processor system, the rule of thumb appears to be to provide $10P$ buffers, but $10P$ buffers may be too expensive for systems containing 10^5 or more processors. Even employing $\Omega(P)$ buffers does not appear to provide any guarantees, however. This paper shows that by organizing the switches so that the messages addressed to a particular processor can use only certain of the buffers, many hotspots can be tolerated with few buffers. For example, a switch with $O(\log P)$ buffers can tolerate a single hotspot with probability 1, and allows the first few hotspots to have a large number of buffers before being declared a hotspot. A switch with B buffers will block a given non-hotspot message with probability less than $O(1/s)$ if there are $O(B/\log s)$ hotspots, and can handle a factor of $O(\ln \ln s)$ more hotspots before the probability becomes a constant. A similar approach can also be used to improve caching behavior in a multithreaded system in which one of the threads tries to consume all of the cache.

Large-scale computing systems typically employ multistage interconnection networks to interconnect their processors. These systems can suffer from ***hotspot contention***, however [7]. Hotspots arise when source processors collectively send too many messages to a particular destination processor—the destination processor falls behind trying to receive messages, and the messages back up into the network filling up buffers in the switches leading to the destination. Then the switches leading to those switches fill up, and eventually the congestion propagates backward through the network, forming a tree rooted at the hot destination of routing nodes with full buffers. This saturation pattern is sometimes called ***tree saturation*** [7]. Hotspots require very little nonuniform traffic and onset can be very fast and can take a long time to alleviate [4]. Data center and supercomputer switches urgently need effective congestion management to avoid performance problems [1].

SPAA'14, June 23–25, 2014, Prague, Czech Republic.
ACM 978-1-4503-2821-0/14/06.
http://dx.doi.org/10.1145/2612669.2612708.

Tree saturation becomes a bigger problem as a system grows to encompass more and more processors. Tree saturation originally was understood as a problem even on networks containing as few as 100 processors [7], but a series of techniques has mitigated tree saturation, at least on networks with as many as thousands of processors. These techniques can be divided into three categories: ***message combining*** [3], ***feedback*** [9] , and ***buffering*** [2, 11]. None of these approaches appear to be in use in modern switches. Combining does not solve the problem for general messages. Feedback is difficult to tune [12]. Here we focus on buffering approaches.

Dias and Kumar [2] do provide a guarantee against hotspots via careful buffer management. Their scheme enforces a rule that each input port may buffer only one message destined to a given processor. They simulated 4 buffers per input port with a 2×2 switch, and found in the face of a single hot spot, the network behaved reasonably well. Although they simulated only one hot spot, their approach can clearly handle more than one hot spot. Given B buffers per input port, a network can tolerate up to B hot spots before non-hotspot traffic is blocked. Our simulation results indicate that it can tolerate $O(B)$ hot spots with good performance, which is not surprising, since it is known that for random routing, only a few buffers are needed per input port.

Today's networks simply employ many buffers, and offer no guarantees in the face of hotspots. For example, Bechtolsheim [1, 6] indicates that for a network containing P processors, a switch should contain enough buffering to hold $10P$ messages. He also points out that the implied memory requirements seem too large to be practical for the next generation of switches.

Using $\Omega(P)$ buffers per switch seems expensive and inefficient. It seems expensive, since with, say 10^6 processors and Ethernet jumbo frames of 10, 240 bytes, each switch requires 10GB of memory for P buffers per switch, or perhaps 100GB if we want $10P$ buffers per switch. It seems inefficient since most of those buffers will not be in use most of the time.

We propose a new buffer management technique called a ***dampening switch***. The idea of a dampening switch is that if there are few hotspots, then each hotspot can use many buffers, but as the number of hotspots increases, the switch reduces the number of buffers that each hotspot may use. This is in contrast to the switch of [2], which permits a hotspot to use only one buffer per switch.

For the purposes of analyzing these switches, we model a hotspot as a processor that has stopped receiving messages. We model the switching network as a store-and-forward network, so that we can think of each in-transit message residing in exactly one buffer. Real hotspots come and go, and real networks employ cut-through routing, but that does not qualitatively change our results. Any buffer containing a message destined to a hotspot is said to be ***claimed*** by that hotspot.

We have designed two kind of dampening switches: ***counting dampeners*** and ***hashed dampeners***. The idea of a counting dampener is to count the number of messages destined to each processor and limit the total number of buffers claimed by by hotspots. The idea of a hashed dampener is restrict each destination processor to a set of buffers using a hash function. In this case, the total number of buffers claimed by a hotspot is limited by the combinatorics of the hashing.

The switch operates by accepting messages across its input links. If messages to particular destination processor exceed their buffer quota, then the switch sends link-level flow control information stopping messages destined to that processor. If all the buffers fill up, then the switch sends link-level flow control stopping all messages.

Dampening Switches

Suppose we design switches containing B buffers such that when there are k hotspots at a switch, we expect $B\alpha^k$ buffers not to be claimed by any of those hotspots (for some $\alpha < 1$). Such a switch design is called a ***dampening switch***. The idea of a dampening switch is that as hotspots appear, each one is allowed to use fewer and fewer buffers. The first hotspots get to use a lot of buffers, but the switch can also handle a lot of hotspots. Since multiple hotspots can appear at once, a dampening switch cannot actually assign a given amount of hotspots to the nth hotspot that appears. Hence we define it in terms of the number or buffers not being claimed by a hotspot.

Dampening switches are appealing because we want to give a hotspot many buffers if there is only one hotspot to give better performance. But we want to be able to handle many hotspots. For example when switches near a processor go through brief periods of heavy traffic, we would like to allocate many buffers to a processor that appears hot. Suppose these switches accept, for example, $B/2$ of those messages as outstanding messages instead of declaring the processor as a hotspot after, for example, 3 of those messages. Then those messages may start clearing up before the traffic jam has a chance to cascade up the network to other switches farther away from the processor. These brief periods of traffic can occur randomly. For example if every processor sends a message to a randomly chosen processor, a standard balls-and-bins argument states that we expect that some processor to receive $\Theta(\log P/\log\log P)$ messages.

Counting Dampeners

The counting dampener method operates as follows. At each switch, we keep a sorted array of how many buffers are being used by each of the processors that have at least one outstanding message at the switch. Each element in the array consists of a processor number and the value by which the list is sorted. This value is referred to as the element's ***value***) We call this list the ***processor impact array***, PIA. Then each time we receive (or transmit) a message from (to) our switch, we update the PIA. If every buffer is full, we stop all new messages from arriving. Otherwise, we find the largest k such that the first k elements in the PIA sum to at least $B(1-\alpha^k)$. Note that k may be zero. We call this k ***the hotspot counter***. We then adjust the flow control so that we block messages destined to all processors in the first k elements.

It turns out that one can maintain the PIA in $O(1)$ time per operation. Calculating the new value of k is also reasonably quick. The arrival or departure of a single message can result in substantial flow-control traffic, however.

Hashed Dampeners

The hashed dampener operates as follows. For each processor, we hash it to a random j-tuple of buffer numbers (that is, integers from 1 to B). We then allow each processor to use only the buffers named in the j-tuple. (We'll discuss how to choose j below.) If all such buffers are full, we block messages destined to that processor. Maintaining the data structure is straightforward, and it turns out that the flow control can be performed by blocking and unblocking $O(1)$ messages per message arrival or departure. As a result, the flow-control overhead can be kept small compared to the message traffic. It also turns out that the protocol, which performs $O(j)$ work per message, can be added to IEEE P802.1p priority-based flow control packet format with very little change. Note that any flow-control mechanism appears to require, in the worst case, $\Omega(j)$ probes into a table since it must look into the j possible buffer locations to find a free location.

To help choose j, the following result is relevant for hashed dampeners.

Theorem 1. *When a hashed dampener has $(B/j)\ln B$ hotspots, we expect $O(1)$ of the buffers to be not in use.*

Proof. It suffices to show that $\log_{B/(B-j)} B \approx (B/j)\ln B$.

$$\begin{aligned}\log_{B/(B-j)} B &= \ln B/\ln(B/(B-j)) \\ &= \ln B/\ln \frac{\frac{B}{j}}{\frac{B}{j}-1} \approx (B/j)\ln B.\end{aligned}$$

□

The analysis of this scheme is similar to analysis for Bloom filters. For example [10] provides a formula estimating the number of elements in a bloom filter of size B with j positions hashed to each element and a given number of bits set which can also be used to reach Theorem 1.

One Hotspot, $O(\log P)$ buffers

We can tolerate a single hotspot with $O(\log P)$ buffers per switch.

The idea is illustrated by this small example. Consider a machine containing 6 processors numbered 0 through 5 with switches containing 4 buffers each, named A, B, C, and D. We allow each processor to use only two switches, as shown in this table:

Processor	Allowed Buffers			
0	A	B		
1	A		C	
2	A			D
3		B	C	
4		B		D
5			C	D

Since $\binom{4}{2} = 6$ we can arrange that each processor has a distinct set of buffers attached to it. Now if any single processor stops receiving, all other processors still have at least one buffer they can use, so their messages continue to make progress.

The probability that a hot spot will block the network falls quickly as the number of buffers increase.

THEOREM 2. *$O(\log(P/\epsilon))$ buffers yields less than ϵ probability of collision.*

Proof sketch: The number of combinations is exponential in the number of buffers, and the probability of collision is less than P^2/C where C is the number of combinations.

Thus, even switches with $o(P)$ storage can provide a guarantee that a single hotspot will not stop messages from being delivered to any other processor, with high probability.

Many Hotspots with High Probability

It turns out that we can tolerate many hotspots if we have a few more buffers. For example, we can tolerate $\Theta(B/\log s)$ hotspots with any particular non-hotspot processor being blocked with probability $O(1/s)$. Or, assigning j buffers to each processor, we can tolerate $(B/j)(\ln(j) - t)$ hotspots with any particular coolspot processor being blocked with probability $1/e^{e^t}$.

Space Sharing

The trick of assigning restricting messages to particular buffers can also address space-sharing in an obliviously-routed butterfly network. In a butterfly network, we can use $O(\sqrt{P})$ buffers to get an interesting space-sharing isolation property. The problem is to divide a machine into disjoint sets of processors that work on different tasks. We want to avoid the messages from one partition from interfering with another. This kind of space sharing isolation was provided, for example, in the Connection Machine CM-5 [5] if the partitions of the machine employed disjoint subtrees of the fat-tree network. In contrast, we can provide a similar property with an arbitrary assignment of processors in a butterfly.

To illustrate how it works, we focus on a binary butterfly. A binary butterfly comprises $P \lg P$ two-input two-output switches. The switches are numbered with pairs (i, j) where $0 \leq i < P$ and $0 < j < \lg P$, where switch (i, j) has outputs connected to the inputs of switch $(i, j+1)$ and $(i \oplus 2^j, j+1)$.

Observe that for messages traveling in the second half of the network, there are at most $\sqrt{P}$ different destinations to which a message can get. So we allocate $\sqrt{P}$ buffers in each node, and assign at least one buffer exclusively to each possible destination. In the first half of the network, there are at most $\sqrt{P}$ different sources that could have gotten to that node, so we assign buffers according to the source

Any message traveling within a partition is guaranteed not to use any buffer used by a message from another partition, because in first half of the network, the message will employ buffers dedicated to the message's source, and in the second half, the message will employ buffers dedicated to the message's destination.

Cache Partitioning

In multithreaded programs one thread can sometimes grab all the cache lines in a shared cache, slowing down other threads. Dampening effectively change the buffer from a fully associative cache to a randomized j-way associative cache. By reducing the associativity, the behavior of the system is improved against processors that receive too many messages. To adapt this idea to solve the cache partitioning problem, we similarly restrict the associativity of the cache. (See [8] for a survey of cache partitioning.)

For caching, there is a clear advantage to allowing a large number of cache lines to be assigned to one thread. Therefore we might adjust the parameters so that, for example, a single uncontended thread can get half the cache.

Acknowledgments

Michael Bender observed that these ideas can be applied to cache partitioning.

This work was supported in part by NSF grants CCF-0937860, CCF-1162148, CNS-1017058, and CCF-1314547.

1. REFERENCES

[1] A. Bechtolsheim. Reinventing datacenter networking. In *HPTS*, Asilomar, Pacific Grove, CA, Sept. 2013.

[2] D. M. Dias and M. Kumar. Preventing congestion in multistage networks in the presense of hotspots. In *ICPP*, volume 1, pages 9–13, Aug. 1989.

[3] A. Gottlieb. An overview of the NYU Ultracomputer project. Ultracomputer Note 100, NYU, July 1986.

[4] M. Kumar and G. F. Pfister. The onset of hot spot contention. In *ICPP*, pages 28–34, 1986.

[5] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the Connection Machine CM-5. *J. Parallel Distrib. Comput.*, 33(2):145–158, 1996.

[6] R. Merrit. Bechtolsheim brainstorms on next networking wave. *EE Times*, Oct. 17 2012.

[7] G. F. Pfister and V. A. Norton. "hot spot" contention and combining in multistage interconnection networks. *IEEE Trans. Comput.*, C-34(10):943–948, Oct. 1985.

[8] D. Sanchez and C. Kozyrakis. Scalable and efficient fine-grained cache partioning with vantage. *IEEE Micro*, 32(3):26–37, May 2012.

[9] S. L. Scott and G. S. Sohi. The use of feedback in multiprocessors and its application to tree saturation control. *IEEE Trans. Parallel Distrib. Syst.*, 1(4):385–398, Oct. 1990.

[10] S. J. Swamidass and P. Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *J. Chem. Inf. Model.*, 47:952–965, 2007. http://www.igb.uci.edu/~pfbaldi/publications/journals/2007/ci600526a.pdf.

[11] Y. Tamir and G. L. Frazier. High-performance multi-queue buffers for VLSI communication switches. In *ISCA*, pages 343–354, Honolulu, HI, 1988.

[12] N.-F. Tzeng. Alleviating the impact of tree saturation on multistage interconnection network performance. *J. Parallel Distrib. Comput.*, 12(2):107–117, June 1991.