Quantifying the difference between scale-out and up

Michael Sevilla

University of California, Santa Cruz

April 29, 2013

Last time... How do we compare scale-up/out?

- How do we choose applications/workloads?
  - → use HiBench[1] [2]
- How do we port applications between architectures?
  - → ≡ methodology, use Phoenix[2] [4, 6, 5]
  - → ≡ functionality, use sequential algorithm

The Plan: port applications, measurements, run on big server

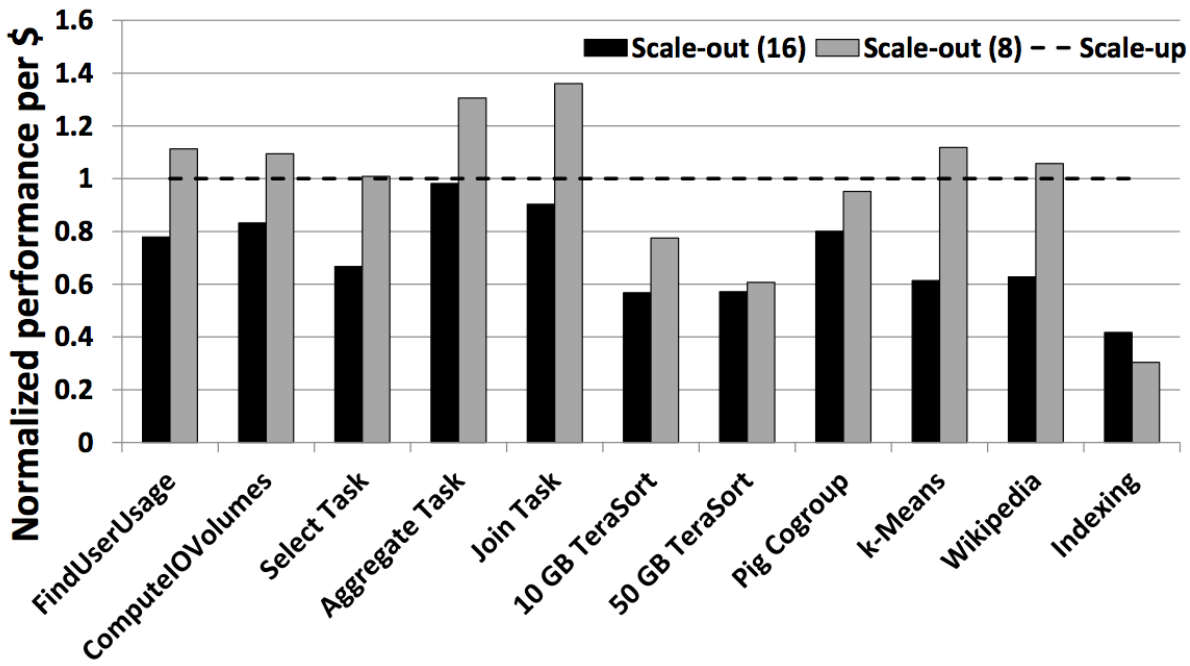
[1] distributed systems benchmark

[2] shared memory MapReduce API/runtime

Related work (1)

Microsoft [1] used performance, $, power, density to judge:

- ▷ *n*-node "Hadoop" vs. 1-node "optimized-for-sup[3] Hadoop"



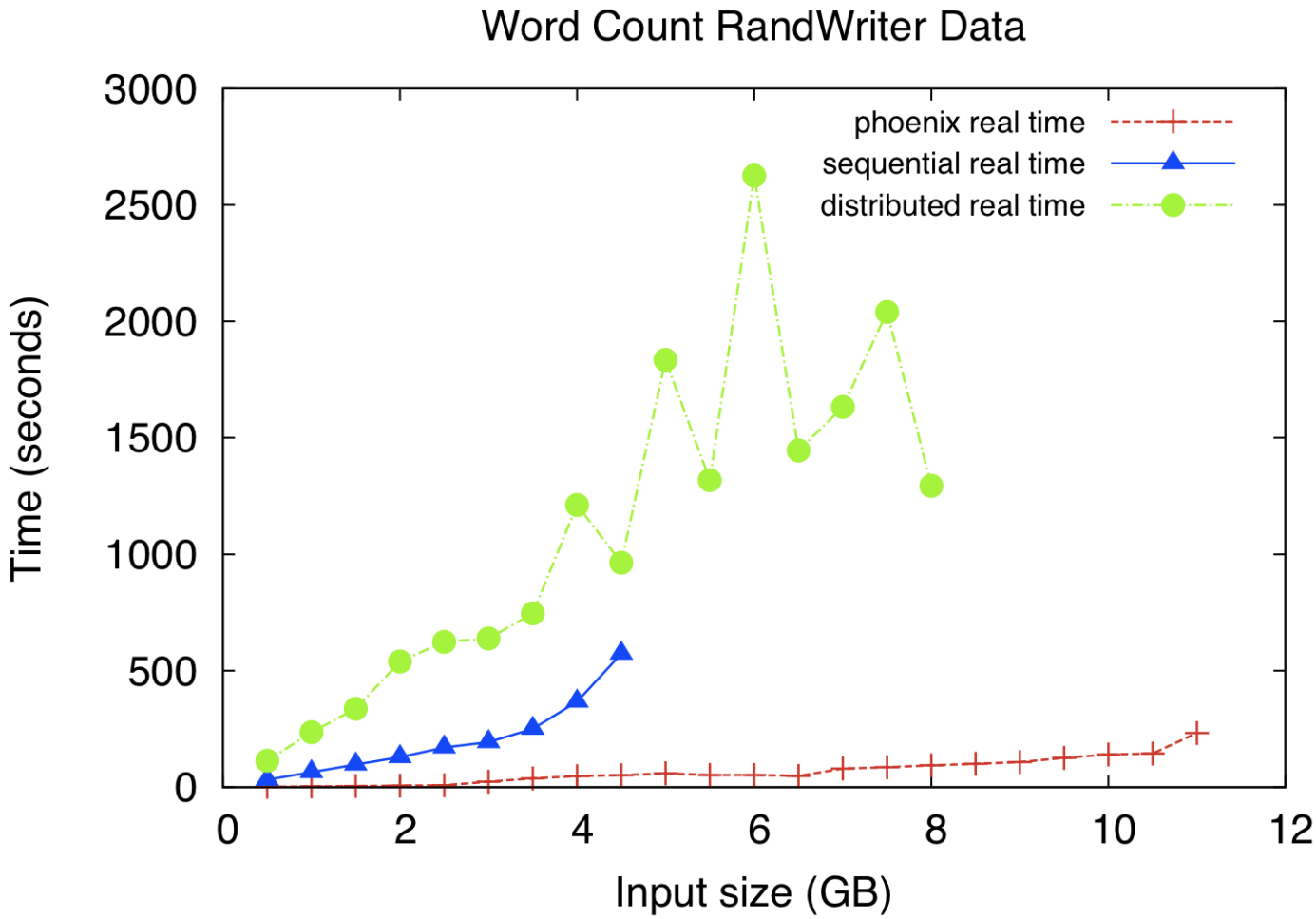
(b) Throughput per $

[3] Removed storage, concurrency, heartbeats, heap size, shuffle

Related work (2)

My work agrees:

- vary machine configurations/data
- profiling, timing, $\frac{\text{mem}}{\text{core}}$ ratios



"s-out programming models are useful for s-up"

"... contrary to conventional wisdom, analytic jobs – in particular MapReduce jobs – are often better served by a scale-up server than a scale-out cluster."

- Microsoft [1], techreport 2012

"... Phoenix leads to scalable performance for both multi-core chips and conventional [SMPs]."

- Ranger [4], HPCA 2007

"[We] show that a scale-out strategy can be the key to good performance even on a scale-up machine."

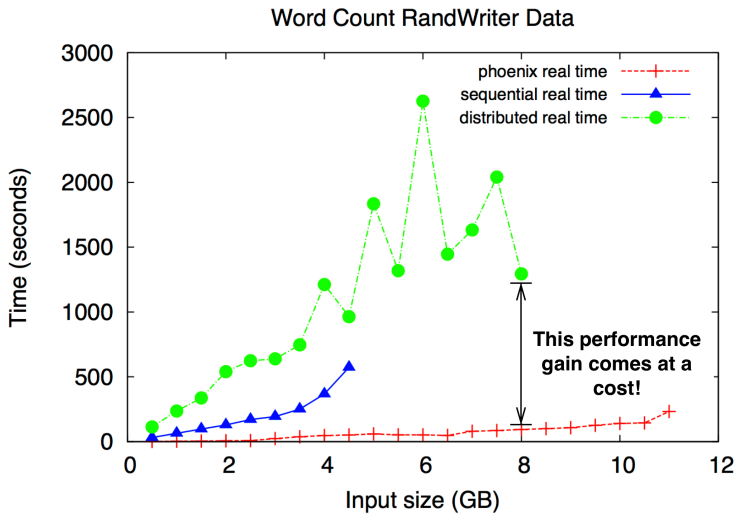
- Michael [3], PDPS 2007

BUT! ... is performance everything?

Why would we choose scale-out?

Because our workload:

- has parallelism
- needs fault tolerance
- needs portability
- needs $>$ storage
- needs $>$ resources
- can run on cheap nodes



Proposal: can we achieve s-out benefits in s-up?

Because our workload:

- has parallelism → Phoenix
- needs fault tolerance → Xen snapshots
- needs portability → HW-aware programming
- needs $>$ storage → hybrid store/compute
- needs $>$ resources → delay "resource wall"
- can run on cheap nodes → cost breakdown

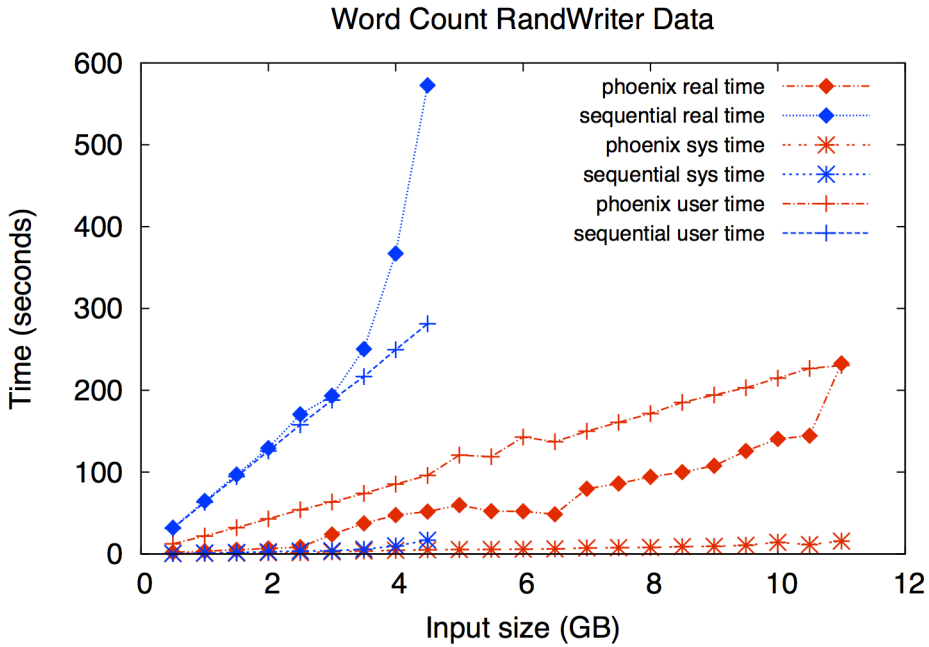
Our contributions:

- ✓ fair/representative way to compare scale-up/out
- ✓ implement "monitor" that achieves above properties in scale-up without incurring overwhelming overhead

Achieving parallel computation...

... using Phoenix

	MapReduce	Phoenix
work distr.	master node worker nodes	parent process threads $\in$ core
communication	network i-keys $\in$ HDFS	shared-memory i-keys $\in$ L1 cache
combiner	$\in$ node after map	$\in$ thread after map



Achieving fault tolerance...

... using Xen, checkpoint state of the **computation**

```
1  checkpoint()
2      while(flag == EXECUTING)
3          // Delete previous snapshot
4          rm ./app.snapshot
5
6          // Leverage Xen's snapshotting
7          xm save ubuntu12-guest ./app.snapshot
8
9          // Set time for snapshot frequency
10         sleep(60)
```

Achieving portability...

... using hardware-aware programming

In a bash script:

```
1  # Get the L2 cache size
2  SIZE=`lscpu | grep cache | grep L2`
3  # Set the value as an environment variable
4  setenv L2_SIZE $SIZE
```

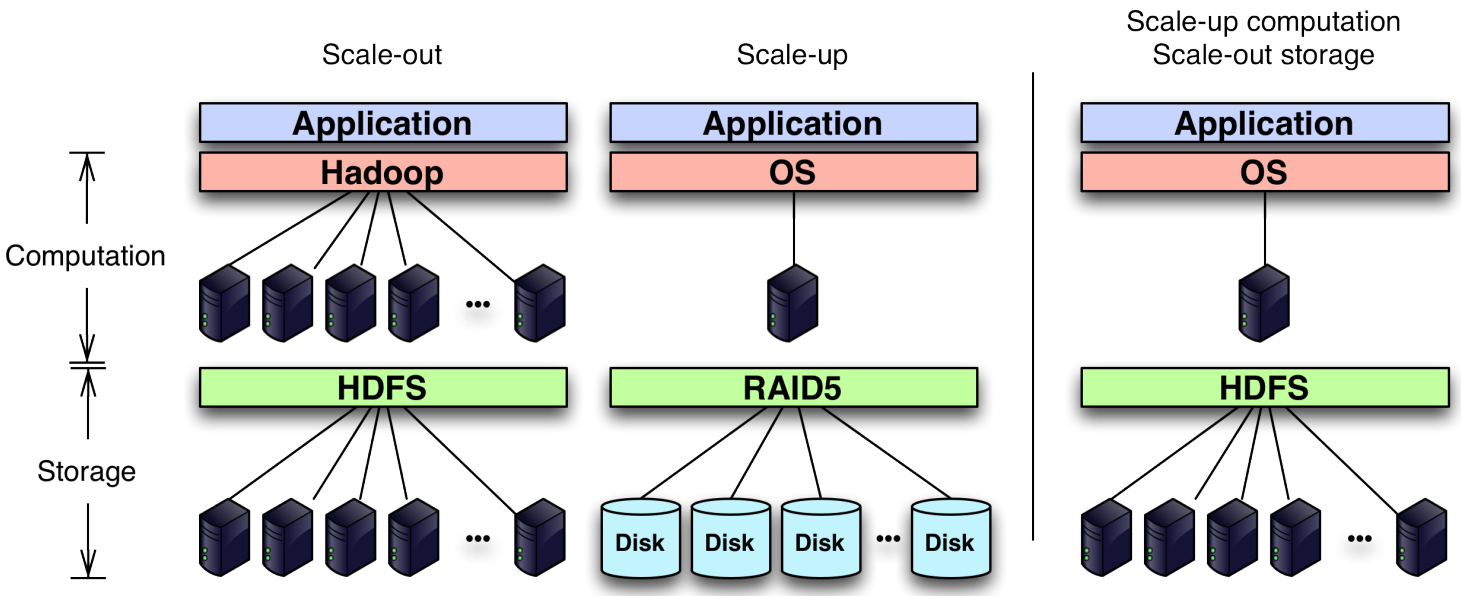
In the application (C++):

```
1  // Get the L2 cache size
2  int l2_size = atoi(getenv(L2_SIZE));
3  ...
4  // Use the value leverage HW configuration
5  block_size = l2_size;
```

Getting > storage...

... using scale-out storage and scale-up computation model

- scale-up storage cannot hold PBs
- scale-up computation < scale-up computation



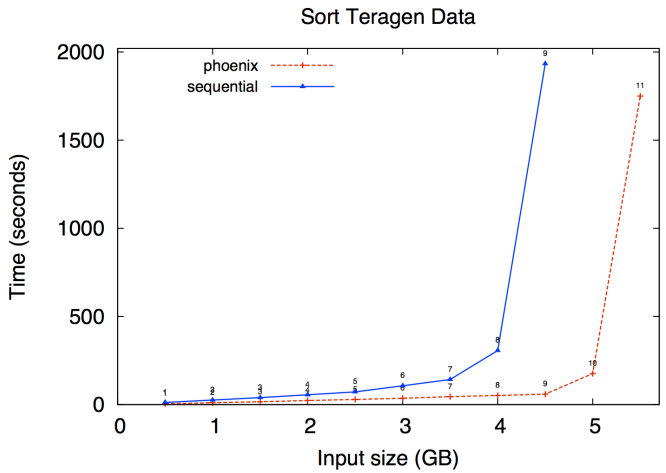
Getting > resources...

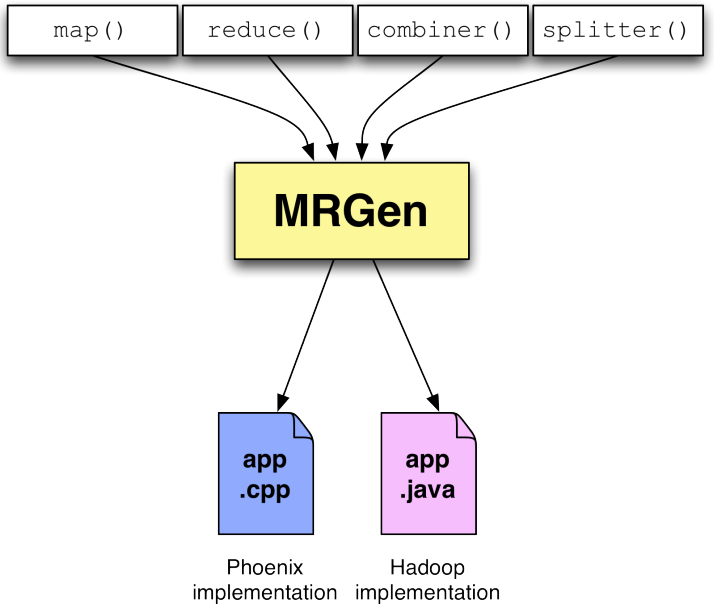
... by delaying the "resource wall"

- using Phoenix (left)
- modifying application data structures

... switching to scale-out

- MRGen - wrapper class that builds both versions (right)





Conclusion

Our contributions:

- ✓ outline benefits we get on scale-out (not in scale-up)
- ✓ fair/representative way to compare scale-up/out
- ✓ implement monitor that achieves above properties in scale-up without incurring overwhelming overhead

Any and all suggestions are welcome. Thanks.

References I

R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron.

Nobody ever got fired for buying a cluster.

Technical report, Mcirosoft Research, Cambrdige, UK, February 2013.

S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang.

The hibench benchmark suite: Characterization of the mapreduce-based data analysis.

In *ICDE Workshops*, pages 41–51, 2010.

M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski.

Scale-up x scale-out: A case study using nutch/lucene.

In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis.

Evaluating mapreduce for multi-core and multiprocessor systems.

In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

J. Talbot, R. M. Yoo, and C. Kozyrakis.

Phoenix++: modular mapreduce for shared-memory systems.

In *Proceedings of the second international workshop on MapReduce and its applications*, MapReduce '11, pages 9–16, New York, NY, USA, 2011. ACM.

R. M. Yoo, A. Romano, and C. Kozyrakis.

Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system.

In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 198–207, Washington, DC, USA, 2009. IEEE Computer Society.

Other ideas

Noah: partition memory/allocate data structures based on

- scale-up
- workload
- NSDI '13 paper

Joe: good idea, he is doing something similar with Hadoop

- use Nathan DeBardeleben's resilience seminars

Dmitris: single-node Hadoop that spawns more works?
Noah:

1. scale up vs. Spark/Tachyon (aggressive mem. caching)
2. Phoenix vs. MPI
3. When does scale-up become limited not by memory but my CPU?
4. How could we scale up Hadoop and then scale out slower?
5. How can we mix scale up nodes with scale out clusters?