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?
  - \equiv methodology, use Phoenix\(^2\) \[4, 6, 5\]
  - \equiv 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”

---

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

![Word Count RandWriter Data](image)
“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
- needs fault tolerance
- needs portability
- needs > storage
- needs > resources
- can run on cheap nodes

→ Phoenix
→ Xen snapshots
→ HW-aware programming
→ hybrid store/compute
→ delay “resource wall”
→ 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

<table>
<thead>
<tr>
<th>work distr.</th>
<th>MapReduce</th>
<th>Phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>master node</td>
<td>worker nodes</td>
<td>parent process</td>
</tr>
<tr>
<td>network</td>
<td>i-keys ∈ HDFS</td>
<td>threads ∈ core</td>
</tr>
<tr>
<td>∈ node after map</td>
<td>i-keys ∈ L1 cache</td>
<td>shared-memory</td>
</tr>
</tbody>
</table>

![Graph showing Word Count RandWriter Data](image)

![Graph showing Time vs. Input size](image)
Achieving fault tolerance...

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

```plaintext
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:

```bash
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++):

```cpp
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-out 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.
Nobody ever got fired for buying a cluster.

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.

Evaluating mapreduce for multi-core and multiprocessor systems.

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.
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?