Comparing Scale-up and Scale-out... 
... an Empirical Study...  

Michael Sevilla  

University of California, Santa Cruz  

March 18, 2013
Q: *What do we do when there is too much data?*

A: *Scale* the system

- **out**
  - ++ nodes to the system
  - → modify applications

- **up**
  - ++ resources to a single node
  - → modify the system

Q: *Which is better?*
Current Trend

1. push towards scale-out (past)

2. difficulty of scale-out (present)

3. push towards scale-up (future?)
Current Trend

1. push towards scale-out  
   ▶ hardware  
     - non-linear scaling  
   ▶ cost  
     - 1 expensive node vs. many commodity servers  
   ▶ interoperability  
     - OSs not designed for ++ resources  
     - Barrellfish, FOS, Corey, Cerberus \[15, 19, 1, 16\]  
     - Linux scalability, LANL study \[2, 3\]

Result: MapReduce, Dryad \[6, 9\]
Current Trend

1. push towards scale-out (past)
   - hardware
   - cost
   - interoperability

2. difficulty of scale-out (present)
   - workload specific architectures
     - Pregel, Spark, S4
   - application optimization
     - concurrent programming
     - parallel databases
     - resource management
   - complexity, unpredictability
     - NFS death spirals
     - faulty network interfaces
     - 100% CPU utilization on the gateway
     - namode/t-trackers/tmp won’t format/start/reset
Maintaining a cluster is not fun

<table>
<thead>
<tr>
<th>No.</th>
<th>User</th>
<th>Time</th>
<th>Issue Description</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Adam Crume</td>
<td>12:04</td>
<td>SSD, bad ram chip</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>12</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD (ssd looks flakey, cannot do an fdisk)</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>13</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>14</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>15</td>
<td>Adam Crume</td>
<td>12:04</td>
<td>can't boot from cd-rom (use usb?)</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>16</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>17</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>18</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>19</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>20</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD issues with 2 hard drives</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>21</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>22</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD: looks okay, keep an eye on it</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>23</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>24</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>25</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>26</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Noah</td>
<td>12:04</td>
<td>SSD                                                                  edid (nomodeset)</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>down</td>
<td>RAM was pulled to fix another host. Replaced RAM is in Joe's desk (at least 1 bad chip in the bunch)</td>
</tr>
<tr>
<td>34</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>35</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>36</td>
<td>No BIOS or POST</td>
<td></td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>37</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>38</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>39</td>
<td>Noah</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>40</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>41</td>
<td>Noah</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>42</td>
<td>Joe Buck</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>43</td>
<td>Michael Sevilla</td>
<td>12:04</td>
<td>keep an eye on this node. Is not booting. Not sure why.</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>44</td>
<td>Noah</td>
<td>12:04</td>
<td></td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>45</td>
<td>Michael Sevilla</td>
<td>12:04</td>
<td>issues SSH key not set</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>46</td>
<td>Noah</td>
<td>12:04</td>
<td>keep an eye on this node</td>
<td>edid (nomodeset)</td>
</tr>
<tr>
<td>47</td>
<td>Joe Buck</td>
<td>12:04</td>
<td>keep an eye on this node</td>
<td>edid (nomodeset)</td>
</tr>
</tbody>
</table>
Current Trend

1. push towards scale-out (past)
   - hardware
   - cost
   - interoperability

2. difficulty of scale-out (present)
   - workload specific architectures
   - application optimization
   - complexity, unpredictability
Current Trend

1. push towards scale-out (past)
   - hardware
   - cost
   - interoperability

2. difficulty of scale-out (present)
   - workload specific architectures
   - application optimization
   - complexity, unpredictability

3. push towards scale-up (future?)
   - simplicity
   - automization
   - evolve
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:
1. narrow methodologies
   - distributed system: number of nodes
     ▶ [15, 1, 16, 19, 13, 20, 17, 18, 2]
   - distributed system: workload types
     ▶ [5, 21, 10]
   - single node: threads/cores
     ▶ [14, 7, 8, 9, 4]
2. out-dated systems
   ▶ (8 × dual-core, 32GB RAM) vs. (14 nodes)
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
   - distr. sys: # of nodes [15, 1, 16, 19, 13, 20, 17, 18, 2]
   - distr. sys: workload types [5, 21, 10]
   - single node: threads/cores [14, 7, 8, 9, 4]

2. out-dated systems
   - (8 × dual-core, 32GB RAM) vs. (14 nodes) [11]

Figure: The POWER5 p5 575 SMP server.
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
2. out-dated systems
   → missing bottlenecks
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
2. out-dated systems
→ missing bottlenecks

► Key Observation: big data uses a lot of data
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
2. out-dated systems
   → missing bottlenecks

- Key Observation: big data uses a lot of data
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
2. out-dated systems

→ missing bottlenecks

▶ Hypothesis: there will be new bottlenecks/slowdowns
Proposal: new scale-up vs. scale-out study

Why? Because previous studies use:

1. narrow methodologies
2. out-dated systems
→ missing bottlenecks

► Methodology: vary machine configs + data
Methodology: vary machine configs + data

Long-term goal: construct performance grid

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$\ldots$</th>
<th>$M_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$p_{11}$</td>
<td>$p_{12}$</td>
<td>$p_{13}$</td>
<td>$\ldots$</td>
<td></td>
</tr>
<tr>
<td>$A_2$</td>
<td>$p_{21}$</td>
<td>$p_{22}$</td>
<td>$p_{23}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_1'$</td>
<td>$p_{11}'$</td>
<td>$p_{12}'$</td>
<td>$p_{13}'$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_2'$</td>
<td>$p_{21}'$</td>
<td>$p_{22}'$</td>
<td>$p_{23}'$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This will help us:

1. create a cost function
2. identify the differences between scaling out and up
Methodology: vary machine configs + data

Short-term goal: small experiment comparing scale-out/up

foreach application
  while(!stressed)
    execute()
    measure_performance()
    ++data

Problem: how do we select applications?
  - representative and feasible
Methodology: vary machine configs + data

Short-term goal: small experiment comparing scale-out/up

```plaintext
foreach application
    while (!stressed)
        execute()
        measure_performance()
        ++data
```

Problem: how do we port applications?

- functionality or methodology?
- fair and feasible
Implementation

Select apps.: existing distr. sys. benchmark (HiBench) [8]
  ▶ word count, sort, Terasort, PageRank, Nutch

Port apps.: Phoenix API/runtime
  ▶ MapReduce → multi-{core, processor}

Evaluating MapReduce for Multicore and Multiprocessor Systems

Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, Christos Kozyrakis
### Phoenix vs. MapReduce

<table>
<thead>
<tr>
<th></th>
<th>MapReduce</th>
<th>Phoenix</th>
</tr>
</thead>
<tbody>
<tr>
<td>work distr.</td>
<td>master node</td>
<td>parent process</td>
</tr>
<tr>
<td></td>
<td>worker nodes</td>
<td>threads ∈ core</td>
</tr>
<tr>
<td></td>
<td>network</td>
<td>shared-memory</td>
</tr>
<tr>
<td></td>
<td>i-keys ∈ HDFS</td>
<td>i-keys ∈ L1 cache</td>
</tr>
<tr>
<td></td>
<td>heartbeat</td>
<td>heartbeat timeout</td>
</tr>
<tr>
<td></td>
<td>remote re-exec.</td>
<td>local re-exec.</td>
</tr>
<tr>
<td>fault tolerance</td>
<td>∈ node after map</td>
<td>∈ thread after map</td>
</tr>
<tr>
<td>combiner</td>
<td>∈ node after map</td>
<td></td>
</tr>
</tbody>
</table>

This makes our comparison:

- ✓ fair
- ✓ feasible
- ✓ representative
## Porting progress

<table>
<thead>
<tr>
<th></th>
<th>scale-out</th>
<th></th>
<th>scale-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop (3 nodes)</td>
<td>≡ methodology</td>
<td>≡ functionality</td>
<td></td>
</tr>
<tr>
<td>✓ WordCount.java</td>
<td>✓ wc.cpp</td>
<td>✓ wc-seq.cpp</td>
<td></td>
</tr>
<tr>
<td>✓ Sort.java</td>
<td>✓ sort.cpp</td>
<td>✓ sort-seq.cpp</td>
<td></td>
</tr>
<tr>
<td>✓ TeraSort.java</td>
<td>X tsort.cpp</td>
<td>X tsort-seq.cpp</td>
<td></td>
</tr>
<tr>
<td>✓ Hama</td>
<td>X pg_rank.cpp</td>
<td>X pg_rank-seq.cpp</td>
<td></td>
</tr>
<tr>
<td>✓ SolrIndex.java</td>
<td>✓ index.cpp</td>
<td>✓ index-seq.cpp</td>
<td></td>
</tr>
</tbody>
</table>
Initial Results: word count

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Time</th>
<th>Error → Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>wc.cpp</td>
<td>11.5 GB</td>
<td>232.82 secs</td>
<td>cpu throttled → int_idle()</td>
</tr>
<tr>
<td>wc-seq.cpp</td>
<td>4.5 GB</td>
<td>572.75 secs</td>
<td>bad allocation → scan_swap()</td>
</tr>
</tbody>
</table>

scale-out vs. scale-up

![Graph showing word count comparison]
Initial Results: word count

<table>
<thead>
<tr>
<th>App</th>
<th>Data (GB)</th>
<th>Time (secs)</th>
<th>Error → Event</th>
<th>+10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>wc.cpp</td>
<td>11.5</td>
<td>232.82</td>
<td>cpu throttled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ int_idle()</td>
<td></td>
</tr>
<tr>
<td>wc-seq.cpp</td>
<td>4.5</td>
<td>572.75</td>
<td>bad allocation</td>
<td>+8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ scan_swap()</td>
<td></td>
</tr>
</tbody>
</table>

scale-up timing breakdown

![Graph showing scale-up timing breakdown]
Initial Results: word count

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Time</th>
<th>Error → Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>wc.cpp</td>
<td>11.5 GB</td>
<td>232.82 secs</td>
<td>cpu throttled → int_idle() +10%</td>
</tr>
<tr>
<td>wc-seq.cpp</td>
<td>4.5 GB</td>
<td>572.75 secs</td>
<td>bad allocation → scan_swap() +8%</td>
</tr>
</tbody>
</table>

scale-up mem/core ratio breakdown

Word Count RandWriter Data

![Graph showing word count over input size for different conditions]
Initial Results: sort

<table>
<thead>
<tr>
<th>Data</th>
<th>Time</th>
<th>Error</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort.cpp</td>
<td>1.5</td>
<td>208.11</td>
<td>OOM; kill</td>
</tr>
<tr>
<td>sort-seq.cpp</td>
<td>4.75</td>
<td>830.46</td>
<td>OOM; kill</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ scan_swap()</td>
</tr>
</tbody>
</table>

Data Time Error → Event

sort.cpp

1.5 208.11 OOM; kill

sort-seq.cpp

4.75 830.46 OOM; kill

→ scan_swap() (+20%)
Initial Results: sort

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Time</th>
<th>Error → Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort.cpp</td>
<td>1.5</td>
<td>208.11</td>
<td>OOM; kill</td>
</tr>
<tr>
<td>sort-seq.cpp</td>
<td>4.75</td>
<td>830.46</td>
<td>OOM; kill; scan_swap() (+20%)</td>
</tr>
</tbody>
</table>
Conclusion

Lays the groundwork for scale-up vs. out study

- choose applications
- port applications
- methodology

The plan:

**Spring Quarter**
- port applications
- profile/take measurements
- write Masters Thesis

**Summer**
- intern @ TidalScale
- hands-on experience

**Fall Quarter**
- document summer experience
- write paper?
Experience with Phoenix

class WordsMR : public MapReduceSort {
    void map(data_types, map_container out) {
        wc_word word = { s.data+start };
        emit_intermediate(out, word, 1);
    }
    ...
    int split(wc_string& out) {
        out.data = data + splitter_pos;
        out.len = end - splitter_pos;
    }
    ...
    bool sort(keyval a, keyval b) {
        return a.val < b.val || ...;
    }
}
...
mapReduce.run()
References I

Corey: an operating system for many cores. 

An analysis of linux scalability to many cores. 

S. S. P. G. Bridges and A. B. Maccabe. 
A framework for analyzing linux system overheads on hpc applications. 

Bigtable: a distributed storage system for structured data. 

The case for evaluating mapreduce performance using workload suites. 
J. Dean and S. Ghemawat.  
Mapreduce: simplified data processing on large clusters. 

Evaluating hadoop for data-intensive scientific operations. 

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. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly.  
Dryad: distributed data-parallel programs from sequential building blocks. 

Pregel: a system for large-scale graph processing. 

M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski.  
Scale-up x scale-out: A case study using nutch/lucene. 
References III

L. Neumeyer, B. Robbins, A. Nair, and A. Kesari.
S4: Distributed stream computing platform.

Evaluating mapreduce for multi-core and multiprocessor systems.

B. Schroeder and G. A. Gibson.
A large-scale study of failures in high-performance computing systems.

A. Schpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, and R. Isaacs.
Embracing diversity in the barrelfish manycore operating system.

A case for scaling applications to many-core with os clustering.

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.

A. Talkington and K. Dixit.
Scaling-up or out.
References IV

D. Wentzlaff and A. Agarwal.
Factored operating systems (fos): the case for a scalable operating system for multicores.

R. M. Yoo, A. Romano, and C. Kozyrakis.
Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system.