A Framework for an In-depth Comparison of Scale-up and Scale-out Systems

Michael Sevilla

Ike Nassi, Kleoni Ioannidou, Scott Brandt, Carlos Maltzahn

{msevilla, inassi, kleoni, scott, carlosm}@soe.ucsc.edu

University of California, Santa Cruz

December 19, 2013
Scaling

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
Scaling

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?*
Contributions

1. Comparison framework for scale-out/up
2. Achieve scale-out properties on scale-up
   - Parallelism limited by new job phases
   - Fault tolerance can make scale-up slower than scale-out
   - Scalable storage may be the ultimate bottleneck

We show:
- must consider properties when comparing scale-out/up
- limitations of a scale-up computation framework
Goal

Framework for comparing:

Why re-examine scale-up?

- new technology
- simplicity
- legacy applications
Goal

Framework for comparing:

Limit study to MapReduce

- standard for big data analytics
- goal is to make fair comparisons
Challenges - How do we:

Q: compare algorithms?
Q: compare hardware?
Q: account for properties provided by scale-out by default?

By design, scale-out provides:
- parallelism by automatically distributing load
- fault tolerance by rescheduling computation
- scalable storage with a distributed file system
- portability; Hadoop applications can run on any cluster
- availability because it can continually service clients
- scalability
Challenges - How do we:

Q: compare algorithms?
Q: compare hardware?
Q: account for properties provided by scale-out by default?

By design, scale-out provides:

- parallelism by automatically distributing load
- fault tolerance by rescheduling computation
- scalable storage with a distributed file system
- portability; Hadoop applications can run on any cluster
- availability because it can continually service clients
- scalability

These properties may or may not affect performance...
...but they can’t be ignored!
Challenges - How do we:

Q: compare algorithms?
Q: compare hardware?
Q: account for properties provided by scale-out by default?

By design, scale-out provides:

- parallelism by automatically distributing load
- fault tolerance by rescheduling computation
- scalable storage with a distributed file system
- portability; Hadoop applications can run on any cluster
- availability because it can continually service clients
- scalability

These properties may or may not affect performance...

...but they can’t be ignored!
Related Work

Scale-up vs. Scale-out Hadoop: Time to Rethink?

ACM Symposium on Cloud Computing ’13 [2, 5]

▶ 10 “typical” jobs
▶ for today’s jobs, scale-up server > scale-out cluster

![Graph showing normalized performance comparison between scale-up and scale-out for various tasks and data sizes.]
Methodology

input

software

hardware

Scale-out

vs.

Scale-up
Methodology

input
► workload, input size

software
►
►
►

hardware
►

→ same workload, scale data

scale-up vs. scale-out

Michael Sevilla

Introduction
contributions
challenges

Methodology
parameters
input
software
hardware

Analysis
initial results
implementations
parallelism

Conclusion
Methodology

input
  ▶ workload, input size

software
  ▶ problem

hardware
  ▶

→ same workload, scale data

→ word count, sort

→

→

→

→ ≡

→ compute contexts

scale-out vs. scale-up

Michael Sevilla

Introduction
  contributions
  challenges

Methodology
  parameters
  input
  software
  hardware

Analysis
  initial results
  implementations
  parallelism

Conclusion
Methodology

input
► workload, input size

software
► problem
► algorithm

hardware

→ same workload, scale data

→ word count, sort
[3]

→ methodology, functionality

Analysis
initial results
implementations
parallelism

Conclusion
Methodology

input
  ▶ workload, input size

software
  ▶ problem
  ▶ algorithm
  ▶ scale-out properties

hardware

→ same workload, scale data

→ word count, sort

→ methodology, functionality

→ implementations

scale-up vs. scale-out

Michael Sevilla
Methodology

input
  ▶ workload, input size

software
  ▶ problem
  ▶ algorithm
  ▶ scale-out properties

hardware
  ▶ processors, memory

→ same workload, scale data
  ▶ word count, sort
  ▶ methodology, functionality
  ▶ implementations

→ ≡ compute contexts
Experimental Setup

Scale-out
- 32 nodes, 2-dual core processors, 8GB RAM

Scale-up
- 1 node, $2 \times 8$-core processors (HT), 256GB RAM
  = 32 compute contexts, 256GB RAM

* node $\in$ scale-out gets the same work as a thread $\in$ scale-up
Scale-up can Perform Better than Scale-out

Word count execution times

![Graph showing the comparison between scale-up and scale-out for Word count execution times. The graph indicates that scale-up performs better than scale-out as the input size increases.](image-url)

- **Introduction**
- **Contributions**
- **Challenges**
- **Methodology**
  - Parameters
  - Input
  - Software
  - Hardware
- **Analysis**
  - Initial results
  - Implementations
  - Parallelism
- **Conclusion**
Scale-up can Perform Better than Scale-out

... but achieving scale-out properties changes the story!

- Other properties must be considered in comparison
Achieving Scale-out Properties on Scale-up

Phoenix: MapReduce runtime for multicore systems \[4, 7, 6\]
→ parallelism, port for methodology

Distributed MultiThreaded Checkpointing (DMTCP) \[1\]
→ fault tolerance

Hadoop Distributed File System (HDFS)
→ scalable storage
Achieving Scale-out Properties on Scale-up

Properties must be considered when comparing scale-out/up
Achieving Scale-out Properties on Scale-up

Properties must be considered when comparing scale-out/up
  - fault tolerance can make scale-up slower than scale-out
Achieving Scale-out Properties on Scale-up

Properties must be considered when comparing scale-out/up

- fault tolerance can make scale-up slower than scale-out
- scalable storage may be the ultimate bottleneck
Achieving Scale-out Properties on Scale-up

Properties must be considered when comparing scale-out/up

- fault tolerance can make scale-up slower than scale-out
- scalable storage may be the ultimate bottleneck
- parallelism limited by new job phases
Achieving Parallelism

- **Introduction**
- **Methodology**
- **Analysis**
- **Conclusion**

### Word count execution times

- **scale-out**
- **sequential s-up**
- **parallel s-up**

### Sort execution times

- **scale-out**
- **sequential s-up**
- **parallel s-up**

---

**Graphs:***

- **Top graph:**
  - X-axis: Input size (GB)
  - Y-axis: Wall clock time (seconds)
  - Lines represent different execution modes.

- **Bottom graph:**
  - X-axis: Input size (GB)
  - Y-axis: Wall clock time (seconds)
  - Lines represent different execution modes.
Achieving Parallelism

Why is sort slower?!
Parallelism is limited by new job phases
  ▶ read: move data from disk into memory
  ▶ merge: sort the data
Sort is slower on scale-up because
  1. new job phases
  2. more key-value pairs
Conclusion

Compare scaling architectures (scale-up/out)
   ▶ comparison framework
      → encompasses \{input, software, hardware\} parameters
   ▶ achieving scale-out properties on scale-up

We show:
   ▶ must consider properties when comparing scale-out/up
   ▶ limitations of a scale-up computation framework
Questions?

Special thanks to our anonymous reviewers for their helpful comments and suggestions.

J. Ansel, K. Arya, and G. Cooperman.
Dmtcp: Transparent checkpointing for cluster computations and the desktop.

Scale-up vs scale-out for hadoop: Time to rethink?

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.

Evaluating mapreduce for multi-core and multiprocessor systems.

Nobody ever got fired for using hadoop on a cluster.

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.