10-20x Faster Software Builds

John Ousterhout

electric cloud

2307 Leghorn Street
Mountain View, CA 94043
www.electric-cloud.com
Slow builds impact almost all medium/large development teams

Electric Cloud speeds up builds 10-20x:
- Harnesses clusters of inexpensive servers
- Unlocks concurrency by deducing dependencies
- Minimizes scalability bottlenecks

Faster builds mean
- Faster time to market
- Higher product quality
- Ability to do more with less
Outline

- The impact of slow builds
- The holy grail: concurrent builds
- Dependencies: problem and solution
- Electric Cloud architecture
- Managing files
- Limiting bottlenecks
- Performance measurements
Problem: Slow Builds

Over 500 companies surveyed, average build 2-4 hours

- **5-15% loss in engineering productivity:**
  - Wasted engineering time & frustration
  - Less time to fix bugs, add features

- **5-10% delay in time to market:**
  - Slow builds add weeks to release cycles
  - Uncertainty & risk due to last-minute broken builds

- **Quality & customer satisfaction:**
  - Developers can’t rebuild before check-in
  - QA waiting on broken builds or skipping tests to meet deadlines
  - More bugs escape to the field
Slow builds drove me crazy

- Sprite research project (Berkeley, late ’80s):
  - Most popular feature was “pmake”
  - Painful to return to commercial OS’es
- Interwoven, 2000-2001:
  - 7-10-hour builds
  - > 1 month with no successful daily builds, late in a release cycle

Discovered that they drive everyone crazy!

Founded Electric Cloud to solve the problem
Theoretical Solution: Concurrency

- Builds have inherent parallelism
- Solution: split up builds and run pieces concurrently
  - Large SMP Machines (gmake –j)
  - Distributed builds (distcc)

*If only it were this easy…*
Problem: Dependencies

- Builds have inherent parallelism
- Solution: split up builds and run pieces concurrently
  - Large SMP Machines (gmake –j)
  - Distributed builds (distcc)
- Current attempts to speed builds yield small results
- Dependency problems:
  - Incomplete
  - Can’t be expressed between Makefiles
  - Result: broken builds

Difficult to get more than a 2-3x speedup

Hard to maintain Makefiles
Deduce dependencies on-the-fly:
- Watch all file accesses: these indicate dependencies
- Automatically detect out-of-order steps

Desired

Actual

Run in parallel? Error!
Electric Cloud Solution

Deduce dependencies on-the-fly:
- Watch all file accesses: these indicate dependencies
- Automatically detect and correct out-of-order steps
- Save discovered dependencies for future builds
- Result: high concurrency possible

**Desired**
Link library \(\rightarrow\) write \(\rightarrow\) x.lib \(\rightarrow\) read \(\rightarrow\) Link app.

**Actual**
Link library \(\rightarrow\) write \(\rightarrow\) x.lib \(\rightarrow\) read \(\rightarrow\) Link app.
\(\times\) Discard \(\times\) Read old
\(\rightarrow\) Link app. \(\rightarrow\) Rerun

\(\times\) Link library \(\rightarrow\) write \(\rightarrow\) x.lib \(\rightarrow\) read \(\rightarrow\) Link app.
Electric Cloud Architecture

- Electric Make
- Cluster Manager
- Node: Electric File System Agent
- Make Machine
- Network

- Plug-in replacement for GNU Make, Microsoft NMAKE
- Inexpensive rack-mounted servers run pieces of build in parallel
- Web-based reporting, management tools
Advantages (vs. multiprocessor):
- Cost-effective: $1-2K per CPU
- Scalable: no hard limit to cluster size

Potential problems:
- Build state not necessarily available on nodes
- Overhead for network communication
- Robustness: more pieces that can break
Virtualization

- Node environment must duplicate make machine; hard because of
  - Different environments on different make machines
  - File versioning within a build
  - ClearCase views

- Simple application-specific network file system:
  - Electric Make is server
  - Agent is client, fetches files on demand
  - Virtualizes subtree(s) from make machine
  - Files cached on nodes during a build

- On Windows, registry data is also virtualized on nodes
Versioning File System

Example: log file extended with series of appends

Files can have many versions during build:
- Append to log file
- Debug/release versions compiled to same .o files

Each read must return correct version (based on **sequential order** for build)

Electric Make maintains version history for each file
- Tricky: name space must be versioned also

Network file system passes appropriate version to each job, flushes caches when necessary
Network Optimization

- P2P file transfers offload 20-25% of outbound traffic:
  - Take advantage of inexpensive bandwidth within switch
- **Just-in-time compression cuts traffic 2.5-3x:**
  - Match network bandwidth to disk
Highly parallel builds stress build machine’s file system:
- Average bandwidth as high as 10-20 MB/s
- ClearCase? High latency

All disk I/O passes through Electric Make: opportunity to manage read & write concurrency
- Single disk? Concurrency causes extra head motion
- Network file system? More concurrency hides network latency

Metadata caching improves ClearCase performance significantly
Recursive Makes

- **Gmake**: separate gmake invocation for each Makefile:
  - Hard to extract & manage concurrency
  - Can’t manage dependencies across Makefile

- **Electric Make**: merge Makefiles
  - Recursive makes return immediately with parameter info
  - Top-level emake manages multiple *make instances*
Recursive Makes, cont’d

- Where this works well:

```bash
all:
  for i in “a b c d e f g”; do \
    cd $$i; $(MAKE); cd ..; \ 
  done
```

- Where this doesn’t work so well (output of submakes is used):

```bash
all:
  for i in “a b c d e f g”; do \
    cd $$i; $(MAKE) >> log; cd ..; \ 
  done
```

- Must modify Makefiles in some cases
Compatibility

Plug-compatible with GNU Make, Microsoft NMAKE:

- Change ‘gmake’ or ‘nmake’ to ‘emake’ in build scripts
- Identical command-line options
- Identical results (except builds run faster)
- Identical log file output
- Typically a few Makefile changes to maximize speedup
Manageability

- Web-based administration
  - As easy to manage many nodes as 1 node
- Can be used by entire team:
  - Supports multiple simultaneous builds
  - Priority system for node allocation
- Robust: automatic fail-over on node failures
### Results: Open Source

<table>
<thead>
<tr>
<th></th>
<th>Local</th>
<th>20 CPUs</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samba</td>
<td>952s</td>
<td>58s</td>
<td>16.4x</td>
</tr>
<tr>
<td>MySQL</td>
<td>1400s</td>
<td>124s</td>
<td>11.3x</td>
</tr>
<tr>
<td>Gtk</td>
<td>891s</td>
<td>95s</td>
<td>9.4x</td>
</tr>
</tbody>
</table>

Diagram: Graph showing speedup of Samba, MySQL, and Gtk with increasing number of CPUs in a cluster.
Results: Linux Kernel

- Linux Kernel 2.6.1
- Make bzimage + modules
- 2.8 GHz Xeon, 1 GB RAM, IDE Drive

<table>
<thead>
<tr>
<th></th>
<th>Build Time [mm:ss]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>22:08</td>
<td></td>
</tr>
<tr>
<td>5 nodes</td>
<td>5:09</td>
<td>4.3x</td>
</tr>
<tr>
<td>10 nodes</td>
<td>2:40</td>
<td>8.3x</td>
</tr>
<tr>
<td>15 nodes*</td>
<td>2:03</td>
<td>10.8x</td>
</tr>
<tr>
<td>20 nodes*</td>
<td>1:42</td>
<td>13.0x</td>
</tr>
</tbody>
</table>

* Projected build time

* Electronic cloud

[Graph showing build time and speedup for different numbers of nodes.]

1328 123 102 160 309 102
Impact: 3 week savings out of an 8 month release cycle expected
Impact: Enabled worldwide follow-the-sun development
We eat our own dog food

Continuous build system:
- Start build and test cycle whenever changes are committed to the main branch
What about distcc?

- Works with gmake –j
- Distributes compile steps to nodes
- Preprocesses code on make machine:
  - Preprocessed code is self-contained: eliminates virtualization issues
distcc vs. Electric Cloud

distcc:
- Free
- Works with other build tools (SCons?)
- Portable
- Compiler-specific (gcc)
- Less scalable:
  - Only distributes compiles; preprocessing centralized
  - Missing dependencies break build
- Build log scrambled
- No cluster sharing facilities?

Electric Cloud:
- Not free
- Only works with Make
- Windows, Linux, Solaris
- Works with all compilers
- More scalable:
  - Distributes all build steps (even Makefile parsing)
  - Deduces dependencies to avoid build breakage
  - Parallelizes sub-makes
- Build log in sequential order
- Cluster mgmt/sharing
Electric Make vs. Distcc

**MySQL**

![Graph showing speedup vs. number of agents for MySQL](image)

**Mozilla**

![Graph showing speedup vs. number of agents for Mozilla](image)

**Apache**

![Graph showing speedup vs. number of agents for Apache](image)

**Linux Kernel**

![Graph showing speedup vs. number of agents for Linux Kernel](image)

- Distcc breaks build
Performance Limits

- File system on make machine
  - ClearCase dynamic views particularly slow
  - Windows: large .pdb and .pch files

- Serialization within builds
  - Linking slow on Linux

- Make machine CPU not an issue
  - Typically running at 30% utilization
## Impact of 10-20x Speedup

<table>
<thead>
<tr>
<th>Build Time</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 hours</td>
<td>Build doesn’t finish overnight</td>
</tr>
<tr>
<td>6 hours</td>
<td>Overnight build</td>
</tr>
<tr>
<td>2 hours</td>
<td>Multiple revs in a single day</td>
</tr>
<tr>
<td>30 min.</td>
<td>Full rebuild before checkin</td>
</tr>
<tr>
<td>5 min.</td>
<td>Little need to switch context</td>
</tr>
<tr>
<td>1 min.</td>
<td>No need to switch context</td>
</tr>
</tbody>
</table>

*Electric Cloud can drop you two bands*
No need to tolerate slow builds anymore

Faster builds mean
- Faster time to market
- Higher quality
- Ability to do more with less
For more information or to answer additional questions:

- Visit our website: [www.electric-cloud.com](http://www.electric-cloud.com)
- E-mail: [info@electric-cloud.com](mailto:info@electric-cloud.com)
- Phone: 650-962-4777