Distributed Software Infrastructure for Scientific Applications

Miron Livny
Center for High Throughput Computing
Morgridge Institute for Research and
University of Wisconsin-Madison

MORGRIDGE INSTITUTE FOR RESEARCH AT THE UNIVERSITY OF WISCONSIN-MADISON

CENTER FOR HIGH THROUGHPUT COMPUTING

THE UNIVERSITY of WISCONSIN MADISON

Disclaimer

I am here to share experience, tools and infrastructure and to offer collaboration - not to present solutions, as developing the methodologies and tools to build cost effective, dependable distributed software is still very much work in progress.

Who are we?

We are part of a Computer Science department (ranked 11th in the US!) and have been working on distributed software tools since the early 80's. So far we failed to engage any other faculty from the department (or other universities) in our software engineering problems/challenges. So all we know and do is self-taught and the result of ongoing experimental work.

Experience

- For more than two decades we have been working on the Condor High Throughput Computing (HTC) software system that has been adopted by a wide range of research and commercial entities.
- For more than a decade we have been leading the Software Area of the Open Science Grid (OSG) which provides a national fabric of Distributed HTC services in the US.

Key points

- Importance of infrastructure - people, tools and computing capacity.
- Complexity of the software supply chain. We are both consumers and producers of software artifacts.
- The risks of new/hot, unproven/emerging and in many cases short lived technologies
- The importance of independent and engaged users

Red Hat Expands Messaging, Realtime and Grid Technology Capabilities to Advance Cloud Leadership

October 14th, 2010 **by Enterprise MRG Team**

Red Hat today announced the availability of Red Hat Enterprise MRG 1.3, including updates to the product's Messaging, Realtime and Grid technologies, which provide a key technology base for Red Hat Cloud Foundations, a solution set that offers a comprehensive set of tools to build and manage a private cloud. Red Hat Enterprise MRG provides an integrated platform for high-performance distributing computing. First released in June 2008, Enterprise MRG has since enabled customers around the world to meet their messaging, realtime and grid computing needs, offering: …

…

Enterprise MRG's Grid functionality, based on the Condor Project created and hosted by the University of Wisconsin, Madison, brings the advantages of flexible deployment to a wide range of applications and workloads.

…

With Grid, customers can build cloud infrastructures to aggregate multiple clouds. It provides integrated support for virtualization and public clouds and easier aggregation of multiple cloud resources into one compute pool. In addition, it provides more streamlined and flexible computing across remote grids with servers, clusters and cycle-harvesting from desktop PCs as well as across private, public and hybrid clouds. MRG Grid is a key base component of Red Hat Cloud Foundations.


redhat
CONGRATULATIONS
DR. MIRON LIVNY AND CHTC TEAM
FIRST RECIPIENT CLOUD LEADERSHIP AWARD


redhat.
2011
Red Hat Cloud Leadership Award
Center for High Throughput Computing
University of Wisconsin Madison

Case 1: 10,000 Cores “Tanuki”

- Run time = 8 hours
- 1.14 compute-years of computing executed every hour
- Cluster Time = 80,000 hours = 9.1 compute years.
- Total run time cost = ~$8,500

- 1250 c1.xlarge ec2 instances (8 cores / 7-GB RAM)
- 10,000 cores, 8.75 TB RAM, 2 PB of disk space
- Weighs in at number 75 of Top 500 SuperComputing list
- Cost to run = ~ $1,060 / hour

Customer Goals

- Genentech: “Examine how proteins bind to each other in research that may lead to medical treatments.”
  - www.networkworld.com
- Customer wants to test the scalability of CycleCloud: “Can we run 10,000 jobs at once?”
- Same workflow would take weeks or months on existing internal infrastructure.

System Components

- Condor (& Workflow)
- Chef
- CycleCloud custom CentOS AMIs
- CycleCloud.com
- AWS

Run Timeline

- 12:35 – 10,000 Jobs submitted and requests for batches cores are initiated
- 12:45 – 2,000 cores acquired
- 1:18 – 10,000 cores acquired
- 9:15 – Cluster shut down

$1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud

By Jon Brodkin | Published 22 days ago

A vendor called Cycle Computing is on a mission to demonstrate the potential of Amazon's cloud by building increasingly large clusters on the Elastic Compute Cloud. Even with Amazon, building a cluster takes some work, but Cycle combines several technologies to ease the process and recently used them to create a 30,000-core cluster running CentOS Linux.

The cluster, announced publicly this week, was created for an unnamed "Top 5 Pharma" customer, and ran for about seven hours at the end of July at a peak cost of $1,279 per hour, including the fees to Amazon and Cycle Computing. The details are impressive: 3,809 compute instances, each with eight cores and 7GB of RAM, for a total of 30,472 cores, 26.7TB of RAM and 2PB (petabytes) of disk space. Security was ensured with HTTPS, SSH and 256-bit AES encryption, and the cluster ran across data centers in three Amazon regions in the United States and Europe. The cluster was dubbed "Nekomata."

Some (Condor) Numbers

Over the past year every month we have:

- Released a new version of Condor to the public
- Performed over 170 commits to the codebase
- Modified over 350 source code files
- Changed over 8.5K lines of code (Condor source code written at UW-Madison as of June 2011 sits at 922K LOC)
- Compiled about 2.5K builds of the code for testing purposes
- Ran 930K regression tests (functional and unit)

Open Science Grid (OSG)
DHTC at the National Level

Some OSG numbers

As we move the Virtual Data Toolkit (VDT) to RPMs, on 10/12/11 we have:

• development	# source RPMs:	186
• development	# 32-bit binary RPMs:	427
• testing	# source RPMs:	179
• testing	# 32-bit binary RPMs:	417
• production	# source RPMs:	24
• production	# 32-bit binary RPMs:	37

Services, Tools

and

Infrastructure

Vulnerability Assessment Service

- Barton Miller
- Jim Kupsch
- Karl Mazurak
- Daniel Crowell
- Wenbin Fang
- Henry Abbey

- Elisa Heymann
- Eduardo Cesar
- Jairo Serrano
- Guifré Ruiz
- Manuel Brugnoli

Vulnerability Assessment of Middleware

- We started by trying to do something simple:

  Increase our confidence in the security of some critical Grid middleware.

- We ended up developing a new manual methodology:

  First Principles Vulnerability Assessment

- We found some serious vulnerabilities … and more vulnerabilities … and more.

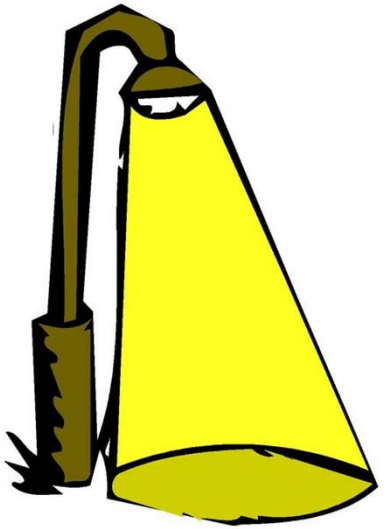
Vulnerability Assessment of Middleware

First Principles Vulnerability Assessment:

- An analyst-centric (manual) assessment process.
- You can't look carefully at every line of code so:

  Don't start with known threats …

  … instead, identify high value assets in the code and work outward to derive threats.



- Start with architectural analysis, then identify key resources and privilege levels, component interactions and trust delegation, then focused component analysis.

Studied Systems

Condor, University of Wisconsin
Batch queuing workload management system
15 vulnerabilities — 600 KLOC of C and C++

SRB, SDSC
Storage Resource Broker - data grid
5 vulnerabilities — 280 KLOC of C

MyProxy, NCSA
Credential Management System
5 vulnerabilities — 25 KLOC of C

glExec, Nikhef
Identity mapping service
5 vulnerabilities — 48 KLOC of C

Gratia Condor Probe, FNAL and Open Science Grid
Feeds Condor Usage into Gratia Accounting System
3 vulnerabilities — 1.7 KLOC of Perl and Bash

Condor Quill, University of Wisconsin
DBMS Storage of Condor Operational and Historical Data
6 vulnerabilities — 7.9 KLOC of C and C++

Studied Systems

Wireshark, wireshark.org
Network Protocol Analyzer
in progress 2400 KLOC of C

Condor Privilege Separation, Univ. of Wisconsin
Restricted Identity Switching Module
21 KLOC of C and C++

VOMS Admin, INFN
Web management interface to VOMS data
35 KLOC of Java and PHP

CrossBroker, Universitat Autònoma de Barcelona
Resource Mgr for Parallel & Interactive Applications
97 KLOC of C++

ARGUS 1.2, HIP, INFN, NIKHEF, SWITCH
gLite Authorization Service
42 KLOC of Java and C

In Progress

VOMS Core INFN
Network Protocol Analyzer
in progress 161 KLOC of Bourne Shell, C++ and C

Google Chrome, Google
Web browser
in progress 2396 KLOC of C and C++

Tools our developers use …

Git, CMake, CPack, Gnu Make, **Coverity**, Metronome, GitTrac, Google coredumper, MySQL to store build/test results, Microsoft Visual Studio 2008 plus Platform SDK, gSoap, valgrind, google-perftools, kcachegrind, DevPartner, gcc, g++, g77, Java, gdb, Perl, Python, GNU tar, rpmbuild, dpkg, gzip & 7zip, patch, lex, yacc, PHP, WiX, CVS, LaTeX, bash, awk, Ruby, gitweb, Cygwin, Ghostscript, latex2html, cfengine, puppet, sed …

Condor & Coverity

- Started using Coverity in 2008
  - First run thousands of "errors"
  - Can take 10 minutes to triage each one
- Strategy:
  - Ignore existing errors. 15 year-old "bugs" can't be that bad
  - Re-run Coverity every release. Aggressively triage and fix all new "bugs" – only ~ 50 new ones to look at
  - Fix original bugs as time permits

Experience

- Over two years, triaged all existing bugs
- Many false positives, but the few bad ones well worth the whole effort
- Ratio roughly 10 to 1 false to real bug
- Trained developers to read Coverity reports and language
- Then new version of Coverity came out. New checkers found new bugs

Use it more frequently!

- Run Coverity on major feature before merge to public branch – Phase II of the transition to IPv6.
- Coverity found three show-stopper bugs which would have taken weeks to diagnose and debug in the field -Fixed these in a couple of hours

Example bug

- Code was changed so that if the DNS server failed on a lookup, a random fd was closed, Coverity pointed out the source code line number of the fault
- Dynamic analysis (valgrind) wouldn't find it as long as DNS server worked
- Fault a long way from failure
- Debuggers would only see the failure

BaTLaB
A Continuous-Integration Facility

Building Communities for SISI Workshop
Arlington, VA - Oct 2011

Todd Tannenbaum

Center for High Throughput Computing

University of Wisconsin-Madison

What: 10,000 foot view

Build and Test Lab = BaTLab

- Lab Infrastructure
  - many different platforms, professionally managed
- Lab Software = Metronome
  - Performs regular builds and/or tests
  - User specifies source location (ex: web server, CVS, SVN, git, …), platforms to use, declares what to build or test
  - Results stored in RDBMS, reports visible via a web portal

Why? Continuous Integration

- Can others outside your environment even build it at all? (*Escrow*)
- Detect problems early
- Ship releases on schedule
- Find problems before users
- Even if code is stable, changes are happening both above and below the application
  - Changes in OS, dependencies, user expectations

Build and TEST!

- Function vs Unit
- Regression tests
- Scalability tests
- “Sweep” tests
- Forward and Backwards compatibility
- Cross versioning

BaTLab Infrastructure

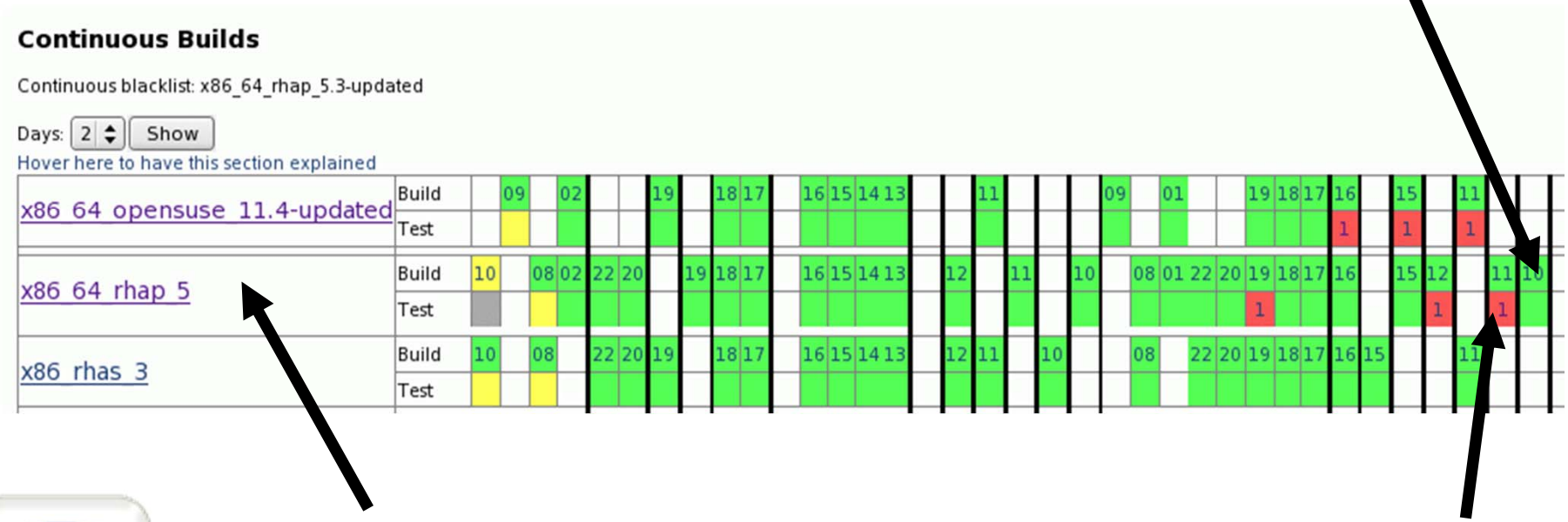
- ~50 unique platforms for builds/tests
- Web portal (http://nmi.cs.wisc.edu)
- 4 submit hosts
- Database cluster
- Backup server
- Network management (DNS, DHCP, SSL)
- Monitoring (Nagios, Ganglia)
- Internal Infrastructure (Condor, …)

Impact on Condor work

- With Batlab, nightly build on all ports
- Bugs found within 24 hours
  - Usually fixed within 24 – 72 hours
    - Still 24 hour latency on all platforms
    - Test failures much harder to debug than build
- Test failures found within 24 hours
  - Unless masked by build failures (problem)
- Developer one-off ‘workspace builds’
  - Much better than before, but still lots of steps

Web portal snapshot

Green build/test here at 10 am



Click here to find out

What happened here?

What happened?

Continuous Build - x86_64_rhap_5 5b630d4bbf6fda4f081ffda665ad9df3182f7c48 Commit info | Log from previous	375395	2011-10-05 11:45:04	PASSED Passed 1 Pending 0 Failed 0	FAILED Passed 0 Pending 0 Failed 1
Continuous Build - x86_64_rhap_5 c6f500cbed80777fdec8a4ca8f4cde8026d5b15d Commit info | Log from previous	375386	2011-10-05 10:45:03	PASSED Passed 1 Pending 0 Failed 0	FAILED Passed 0 Pending 0 Failed 1
Continuous Build - x86_64_rhap_5 a8b91be0756ba4194d7e32131340540[illegible]ca888bd Commit info | Log from previous	375376	2011-10-05 08:47:45	PASSED Passed 1 Pending 0 Failed 0	PASSED Passed 1 Pending 0 Failed 0

Click here

Whom to blame?

projects / condor.git / log

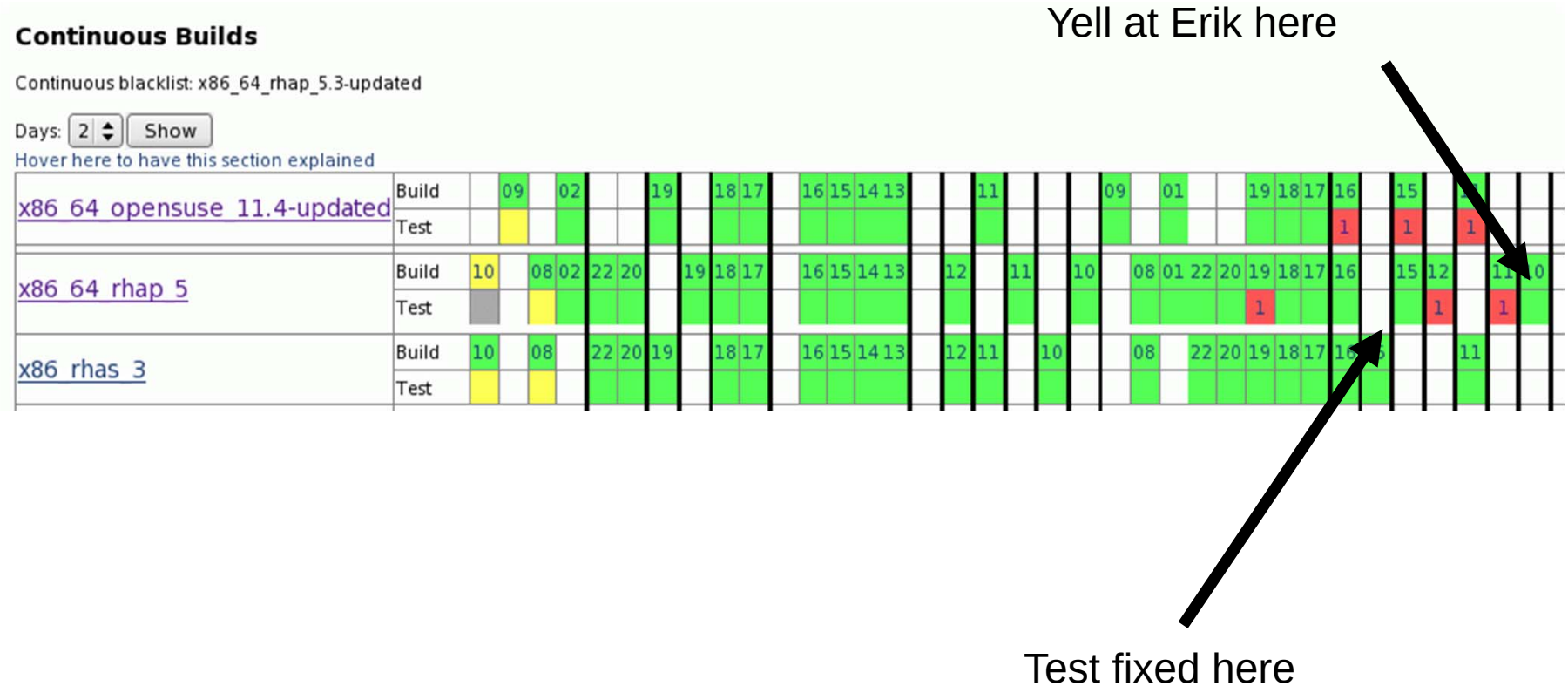
summary | shortlog | log | commit | commitdiff | tree
first · prev · next

condor.git

2 days ago	**===VersionHistory:Completed=== ===GT=== #2514**
commit | commitdiff | tree	*Erik Erlandson [Wed, 5 Oct 2011 15:30:00 +0000]*
	===VersionHistory:Completed=== ===GT=== #2514
2 days ago	**Added a regression test for basic partitionable slot capability ===GT:Fixed=== #2514**
commit | commitdiff | tree	*Erik Erlandson [Wed, 5 Oct 2011 15:26:29 +0000]*
	Added a regression test for basic partitionable slot capability ===GT:Fixed=== #2514
2 days ago	**add cygwin\bin to the front of the path in remote_pre for windows**
commit | commitdiff | tree	*John (TJ) Knoeller [Wed, 5 Oct 2011 15:11:11 +0000]*
	add cygwin\bin to the front of the path in remote_pre for windows tests in NMI to force cygwin perl to be used to run batch_test.pl. ===VersionHistory:None===

Condor is a specialized workload management system for compute-intensive jobs.

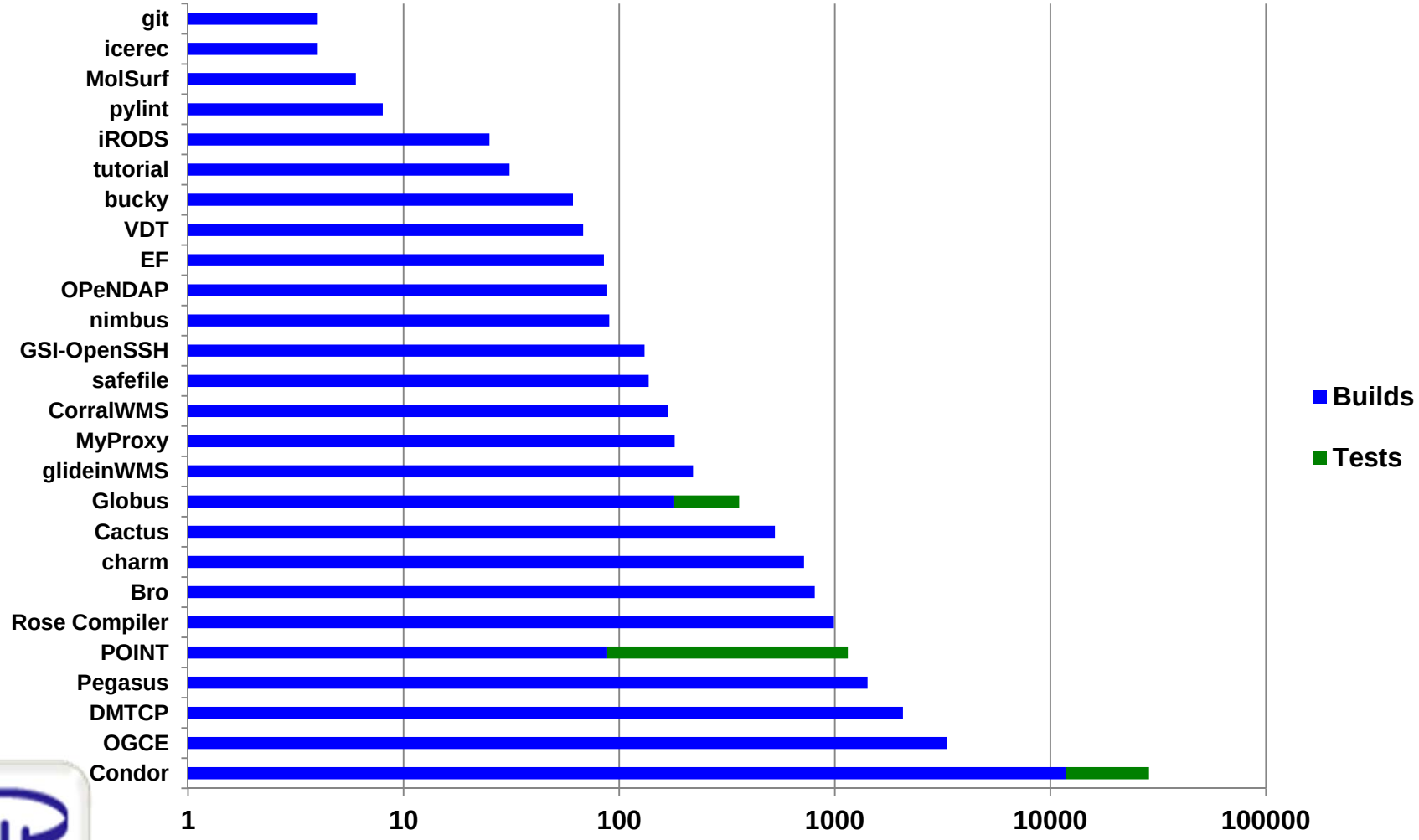
Back in business



“Hourly builds” on three platforms

- Builds and esp tests fall behind
  - Soln: JobPrio == Qdate
- Dramatically improved # of green nightly builds – almost always, except for late pushes
  - Lesson learned – more build per day, better

Usage by Project, last 90 days



Adding software to the VDT

1. Decide what software is needed
2. Intake
3. Prepare
4. Internal testing
5. Integration Testbed
6. Release
7. Support

Open Science Grid

Step 3: Prepare

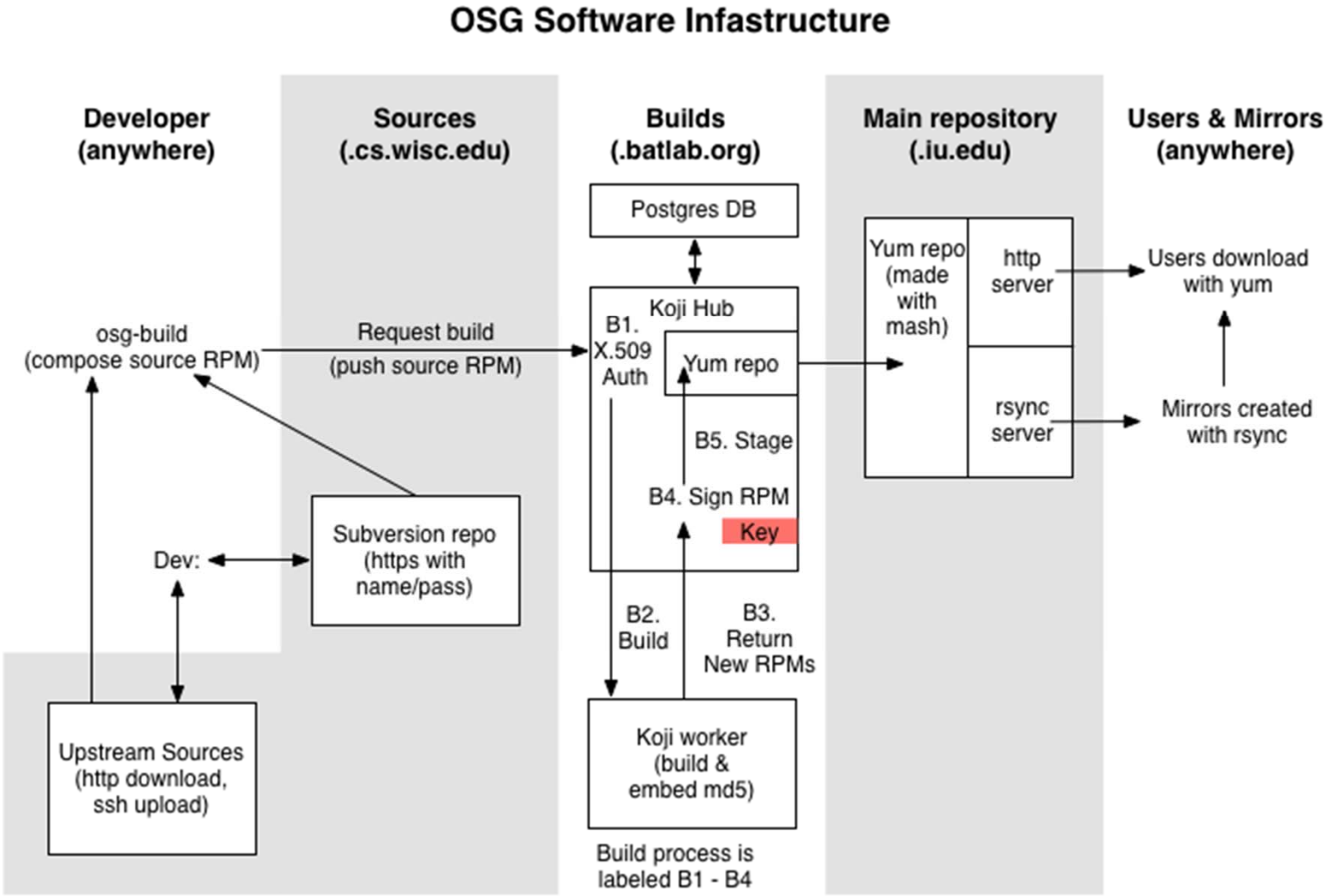
Option 1: We do all the work

- Package & build
- Provide configuration
- Test
- Document

Option 2: Borrow from the community

- If the software is already packaged appropriately, use it.
- May still need to provide configuration
- Still need to test
- Still need to document

Building & Distributing the VDT



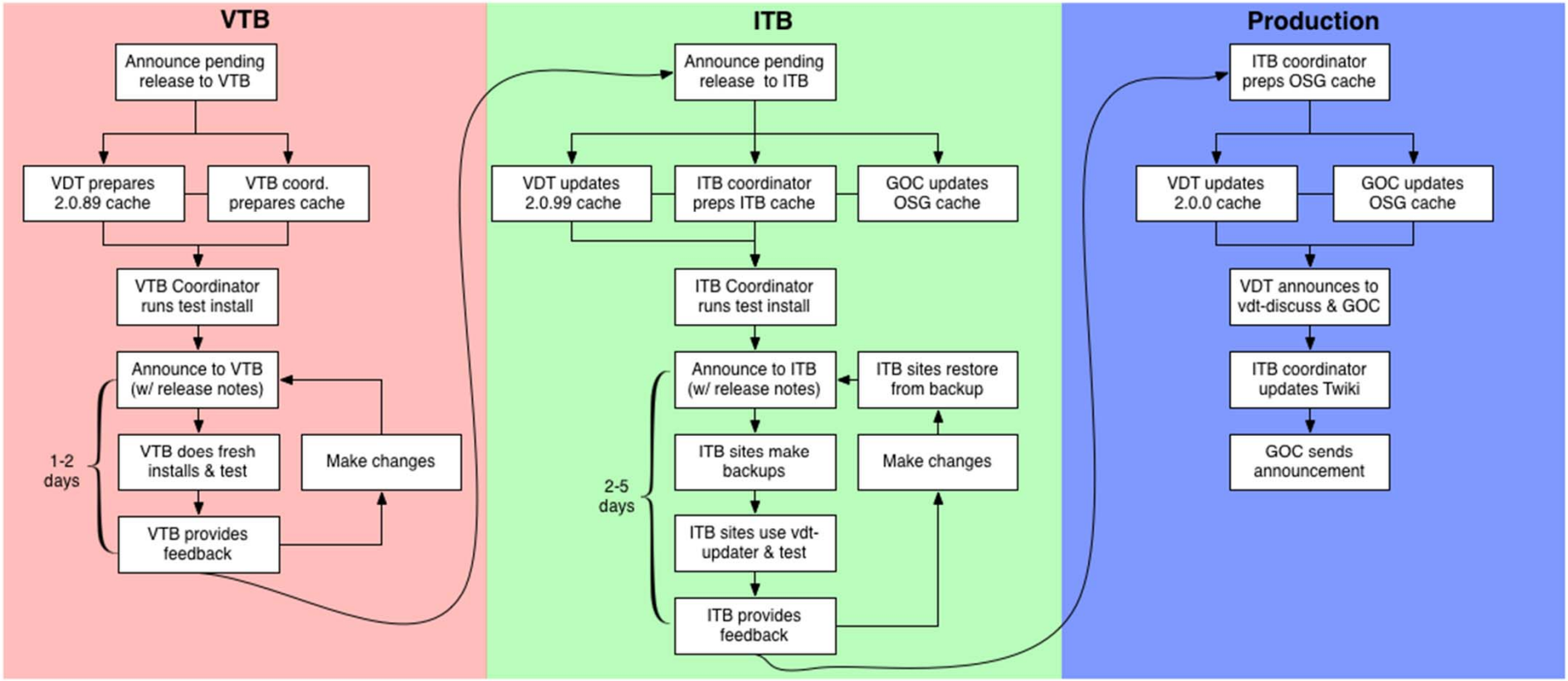
Step 4: Internal Testing

- Daily testing is essential
  - Reports to developers
  - Test against:
    - All supported operating systems
    - Pre-releases of operating systems (find out problems before they strike)

Step 5: Integration Testbed

- Wide-area OSG testbed with real-world (i.e. not developer) environments
- Verify installation process
- Run appropriate tests
  - Small updates require basic tests
  - Large updates require participation from users to ensure their scientific workflows still work

Testing and Deployment



We are missing a forum to discuss challenges, share experiences, talk about failures and report successes