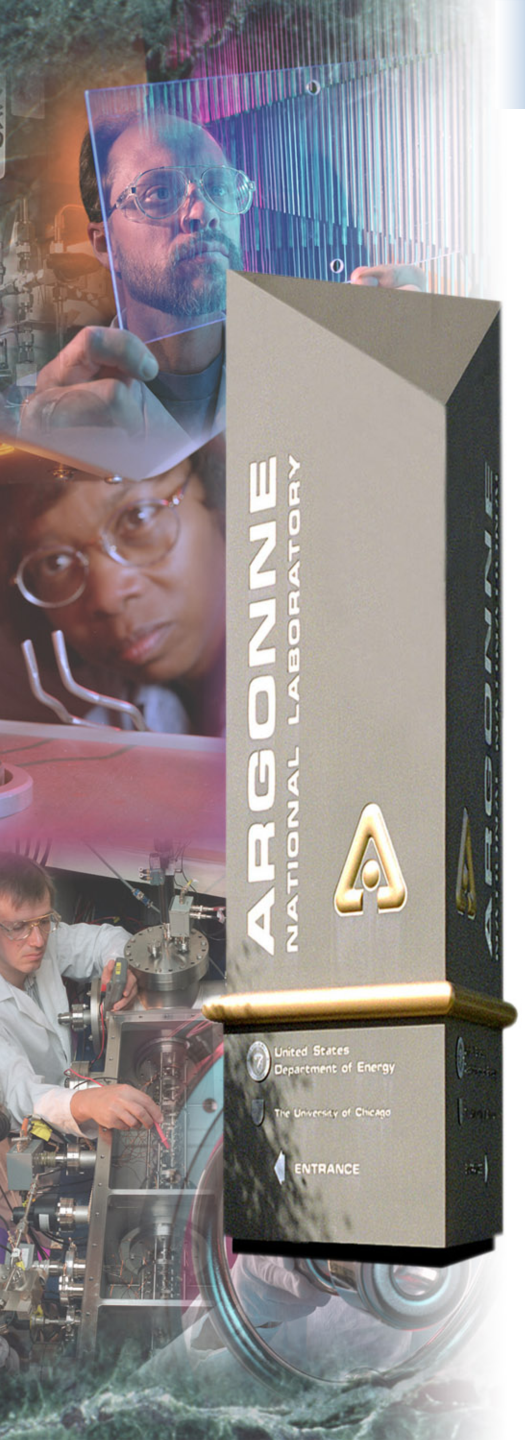


ARGONNE NATIONAL LABORATORY

Bcfg2: A Pay as You Go Approach to Configuration Complexity

AUUG 2005

Narayan Desai
desai@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory
October 21, 2005

THE UNIVERSITY OF CHICAGO

Office of Science
U.S. DEPARTMENT OF ENERGY

Overview

- A Bird's Eye View of Configuration Management
- The Adoption Problem
- Our Initial Observations (and some history)
- Bcfg2
- The Pay as You Go Addition
- Experiences
- Conclusions
- Status and Future Work

Configuration Management

- A research area consisting of several related tasks
  - the mechanics of system configuration deployment
  - the creation of client configuration specifications
  - specification/constraint languages
  - autonomic processing
- Management research has occurred on Unix systems for quite some time
- Though quite immature
  - no common consensus about problem framing or methods
- Motivations
  - System size and complexity continues to grow
  - Available administrator manpower is stagnant
  - External importance of computers and their services continues to grow
  - Tools stand to provide major benefits to real systems
- More than cfengine

(My view of) Goals of Configuration Management Tools

- Ease the system configuration process
  - Leverage uniformity
  - Provide constructs for architecture and function abstraction
  - Expose a declarative specification for systems
    - *Describe goals, not actions*
  - Remove the repetitive tasks from administration
- Create an administrative application portability framework
- Provide an accurate assessment of current network configuration states
  - Divergence from the central specification
    - *Incorrect configuration*
    - *Extra configuration*
- Make system configuration processes proactive, not reactive

System Administration Myths

- Each of my systems is unique.
- I will never need another system like this.
- Manual system patching is good enough.
- I can maintain uniform configurations using manual system configuration processes.
- My configuration doesn't change very often.
- System configuration tools make my skills less important.
- This system is a temporary solution.

Widely Deployed Configuration Management Tools

- cfengine
- SystemImager
- KickStart
- JumpStart
- AutoYAST
- Redhat Network
- dd
- *insert your favorite build automation program here*

Shortcomings of Deployed Tools

- Many tools only support system installation
  - Incremental configuration changes must be tracked with another tool
- Some tools don't support reasonable sorts of system differentiation (particularly imaging tools)
- Imperative
- Not generally proscriptive
- Architecture specific
- Don't provide high-level constructs
- Freaky

Complex/Research Tools

- LCFG
- Quattor
- Arusha
- Puppet
- Bcfg2
- Ed Smith's constraint systems
- Sanity
- SmartFrog
- Akamai Configuration Propagation System
- IsConf 2/3

Tool Capabilities

- Coherent reconfiguration
- Improved user models
- Declarative syntax
- Basic autonomics
- High-granularity parameterized configurations
- Support for collaborative system administration

The Adoption Problem

- Despite attractive features, these tools remain unused
- Lack of userbase takes a toll
  - Tools remain site-specific
  - Lack of user feedback hinders research
- Administrators manage systems less efficiently
- Usability (and perception thereof) is likely a large factor
  - Chicken and Egg problem
  - Tools are too hard to deploy
  - User interface
  - Benefits unclear
- Lose – Lose Situation
  - Tools suffer
  - Users suffer

Some History (and Observations)

- MCS has a long involvement with configuration management (Remy Evard)
  - cfg-get (1995)
  - site configuration survey (1997)
  - sanity (1998)
  - culture of methodical system management
- We still encountered deployment stalls
  - Some local administrators fought against use of sanity
  - Despite unversal buy-in to “configuration management”
- Usability problems
  - Unfamiliar model
  - Insufficient benefits to justify large-scale methodology changes
  - Tools required uncanny knowledge of the entire environment
- Some Administrators just punted and manually managed systems

More Recent History

- Bcfg development started in 2002
- Primarily developed to address usability shortcomings
- Initial work indicated that “one true user interface” wouldn't work
  - Different problem solving styles
  - Different goals
  - Variety of patterns in configuration data
- This motivated our incremental approach to configuration complexity
- Earlier this year, we completed production deployment of Bcfg2
- This has provided much more insight about the scope of this problem

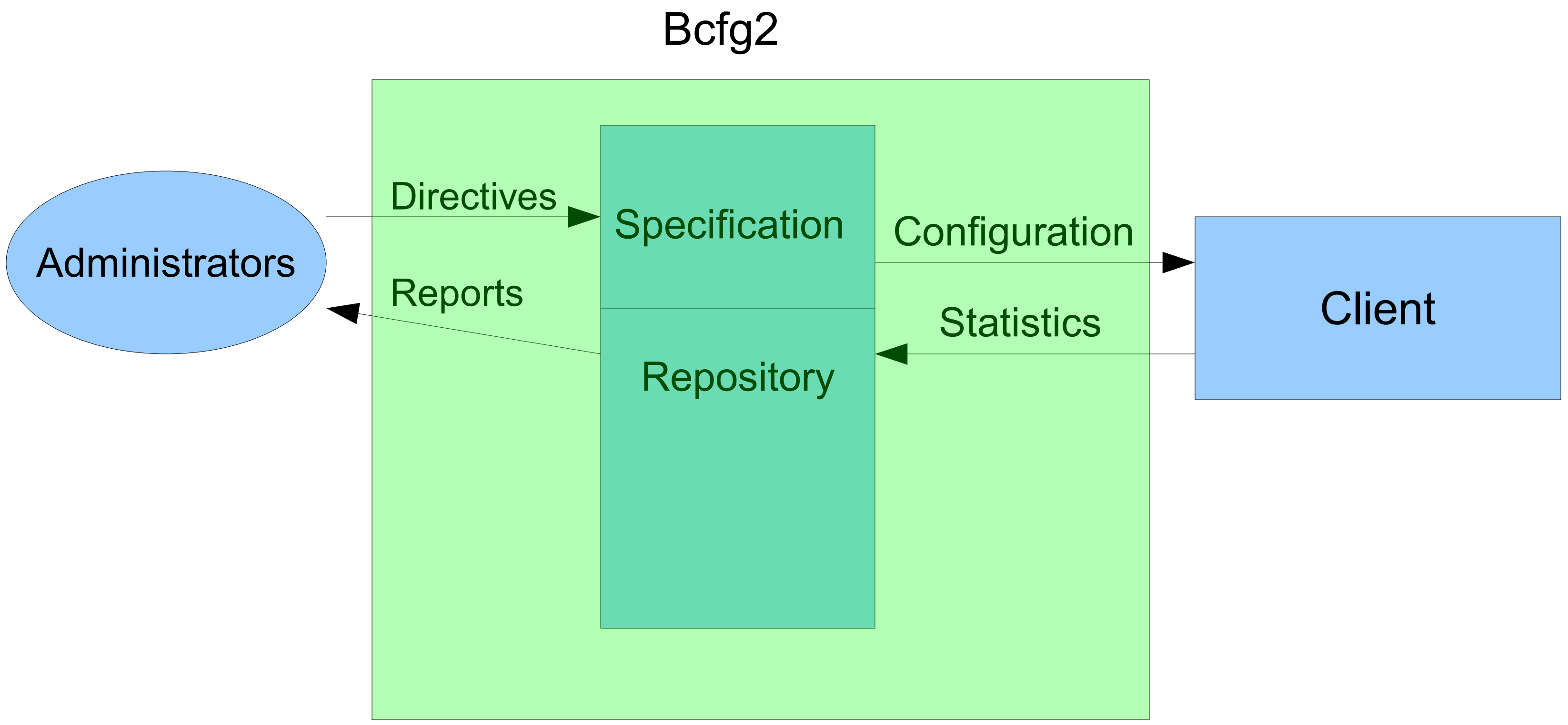
Bcfg2 Overview

- Client/Server configuration management tool
- Provides abstract classing mechanisms
- Includes architecture abstraction
- Implemented in python
- Currently supports Linux (Redhat, Suse, and Debian) and Solaris
  - Adding platform support is trivial (~100 lines of code)

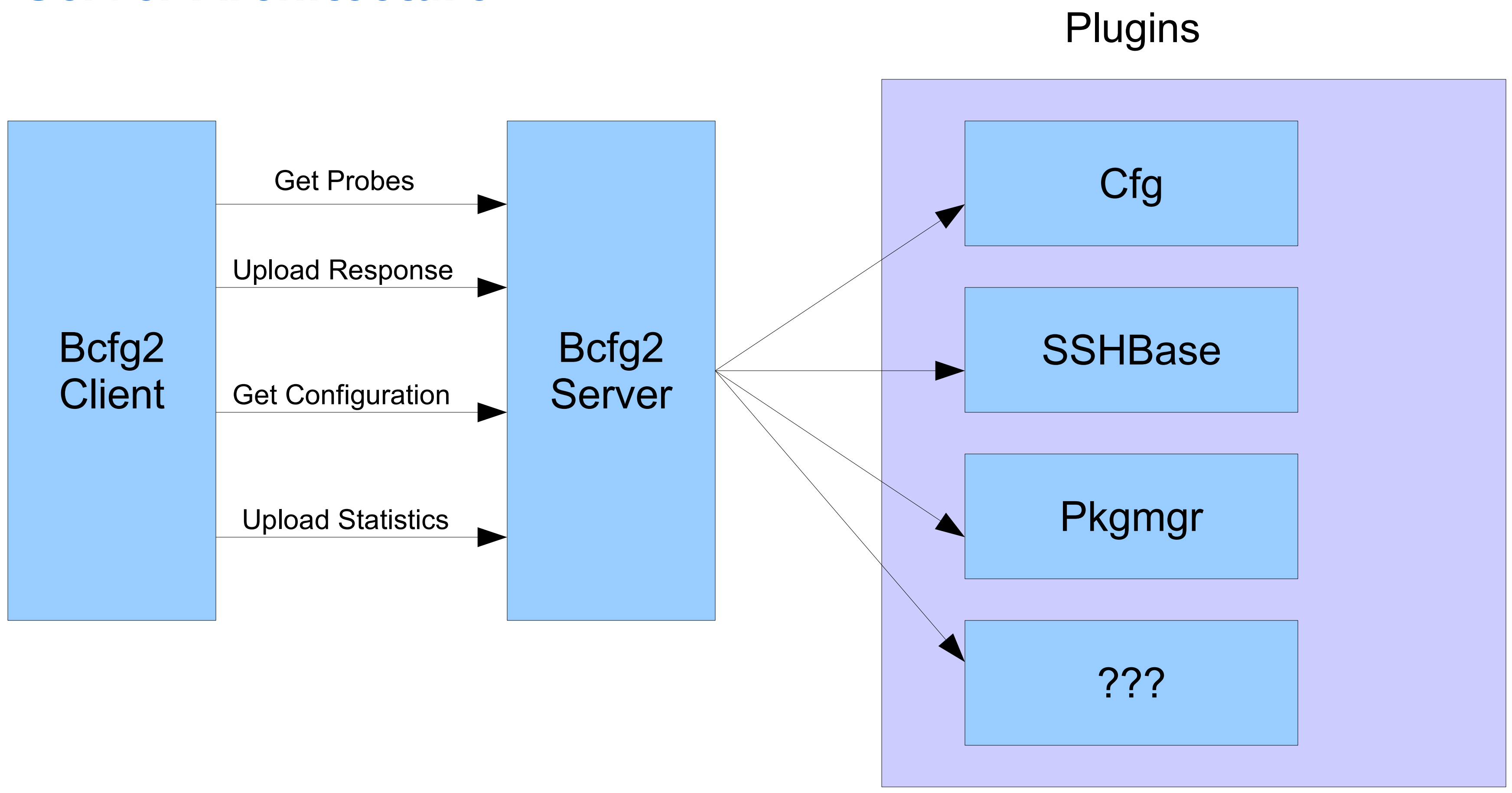
Bcfg2 Design Goals

- All configuration is driven from a central specification
  - Serialized into a per-client view
- The specification is proscriptive
  - No client side “blind spots”
- Client configuration state information must be readily available to administrators
- Administrators interact with Bcfg2, not individual clients
- Server extensibility is essential
  - Single model problem
  - External canonical data
- Keep the client simple
  - Complicated logic is more easily managed on a server
- Provide declarative specification layer
- Attempt to strike right balance between simplicity and flexibility

Bcfg2 Architecture



Server Architecture



Plugins

- Server-side extensions that contribute to the client configuration process
  - Probe client-side data
  - Provide configuration inventory
  - Provide configuration entry contents
- Loadable at runtime
- Must implement an interface
- Arbitrary internal logic is expected
  - Can use a domain-specific data representation for brevity and clarity

The Initial Four

- Cfg
  - A configuration file repository
  - Files are tagged with a metadata attribute
  - Most specific file wins
- Pkgmgr
  - Per-image package inventory
- Svcmgr
  - Provides class-based access to service activation
- SSHBase
  - Manages ssh host keys and known hosts file
  - Centrally run
  - Allows keys to persist through a rebuild
  - Allows central revocation of ssh host keys
  - Coordinates a consistent, correct ssh_known_hosts file

The Initial Four (cont)

- All use literal representation
  - Configuration files in a directory hierarchy
  - Lists of package versions
  - Directives for service activation
- Each provides an intuitive model
  - Users quickly become comfortable with each of these
  - Initial setup is quite straighforward
- Ease here aids in adoption
  - Users can easily envision what the tool is good for
- While systems can be run in this way, more advanced models become appealing
  - Enter the plugin API

The Pay as You Go Approach

- Some situations call for more complex representations
  - parameterized configurations
  - complicated workflows
- Most systems provide a complex language to support these operations
- Bcfg2 takes a more lightweight approach
- Default plugins provide a very literal interface
  - Files on the filesystem
  - Simple transformation rules
- More abstract plugins can be easily implemented and enabled
  - Domain-specific languages can be used to describe important systems
  - Complex workflows can be automated
- Due to the plugin interface, configuration that doesn't require complex logic can still be represented as opaque blogs of file data

Abstract Representations

- Cheetah
  - Templating plugin based on the Cheetah engine
  - Raw templating can be quite foreign to users
- Task-specific Plugins
  - Host Management
  - Webserver Management
  - User Management
  - Other tasks

ARGONNE NATIONAL LABORATORY

Task-Specific Plugins

- Analogous to “Administrative Applications” mentioned previously
- Substantially better than standalone systems
  - Can use the configuration management system for all of the grungy details
    - *performance of configuration changes*
    - *central data repository*
    - *access to other specification data*
  - The generation target moves from imperative logic to declarative specification
    - *The admin app only needs to describe what end state is desired*
  - These two factors strip out many of the factors that make these applications non-portable

Deployment Overview

- Main deployment process occurred from December 2004 – April 2005
- Progress by machine categories
  - Clusters
  - Workstation environment
  - Servers
    - *Most complicated*
    - *Still not quite complete*
- The design of Bcfg2 was substantially altered during this process
  - More emphasis on reporting
  - Driven by administrator needs/complaints

Complexity Issues

- Administrators try to KISS when possible
  - Particularly with new tools
  - Particularly on servers
  - and so forth
- Sophisticated solutions are not initially attractive
  - Once confidence in the tool develops, users are ready to try more complicated things
  - Using these initially tends to be a non-starter
- In large groups, a range of needs is evident
  - Different tasks require different configuration patterns
  - The cost of manual actions is quite different
    - *Clusters*
    - *Workstations*
    - *Servers*
- As users become more comfortable with tools, their assessment of utility changes
  - To the point that different users can't communicate effectively

Experiences

- The deployment process was like herding cats
  - Different administrators got comfortable with Bcfg2 at different rates
  - This meant that a large range of comfort in Bcfg2 existed throughout the process
- The incremental approach to complexity helped substantially
  - Administrators started with very literal representations of the configuration
  - These literal representations made the tool very predictable
  - Aided in the scratch and sniff test
- After the basic configuration issues were addressed a second pass for complex configuration processes was made
  - We focused on time-consuming aspects of system reconfiguration
    - *Web server configuration*
    - *Host management*
    - *User management*

Conclusions

- The user model question is far from answered
- Supporting a variety of methods and representations is the only way to make progress
  - So that tools are usable in a variety of environments
  - So that users find tools intuitive
- Adaptive approaches allow a reasoned decision about desired complexity level to be made
  - You can change your mind
  - Initial focus is major issues
  - Optimization comes later
- Bcfg2 is worth looking at
  - I might as well come out and say it
  - It is designed as a general tool
  - If it won't work for you, we would love to hear why

Status

- Bcfg2 is publically released
- In use outside of ANL
- Documentation exists, and is constantly being improved
- User feedback is constantly resulting in model and feature improvements
- We need more feedback

Future Work

- While this design was practical much work remains
- The data store needs improvements
  - Current plugins have discrete repositories
  - No coherent data overlap is possible
  - Everything is interrelated
- Plugins need some sort of user interface
  - Network Transparency
  - Authentication
- A generic approval and delegation is needed for several plugins
- Inter-server synchronization

Questions

- ???

- http://www.mcs.anl.gov/cobalt/bcfg2/
  - Documentation
  - Papers and Presentation
  - Code
  - Mailing list archives