PyCon
Connecting The Python Community

COMPETE, INC.
The Predictive Edge

CFS: The Compete File System

File System Virtualization using Python

Christopher Gillett
Chief Software Architect
Compete, Inc.

Copyright © 2000-2005 Compete™ All Rights Reserved.

Background

- Compete, Inc. analyzes large amounts of data (gigabytes per day), and accrues terabytes of data every year supporting its predictive analysis business

- Technology platform is Unix clump/cluster/grid of 60+ machines
  - Job level parallelism managed using Portable Batch System
  - Storage is NFS with dedicated NFS servers
  - Platform has been evolving for a few years
    - Lots of storage devices with differing sizes
  - Software developed in-house in a variety of languages:
    - C++, C, Java, Python, etc.

- Problem: How to effectively manage lots of data on lots of different devices
  - Sub Issue: Small development staff
  - Sub Issue: Even smaller budget
  - Sub Issue: Needs to integrate cleanly into existing software, scripts, etc. with little/no work

Copyright © 2000-2005 Compete™ All Rights Reserved.

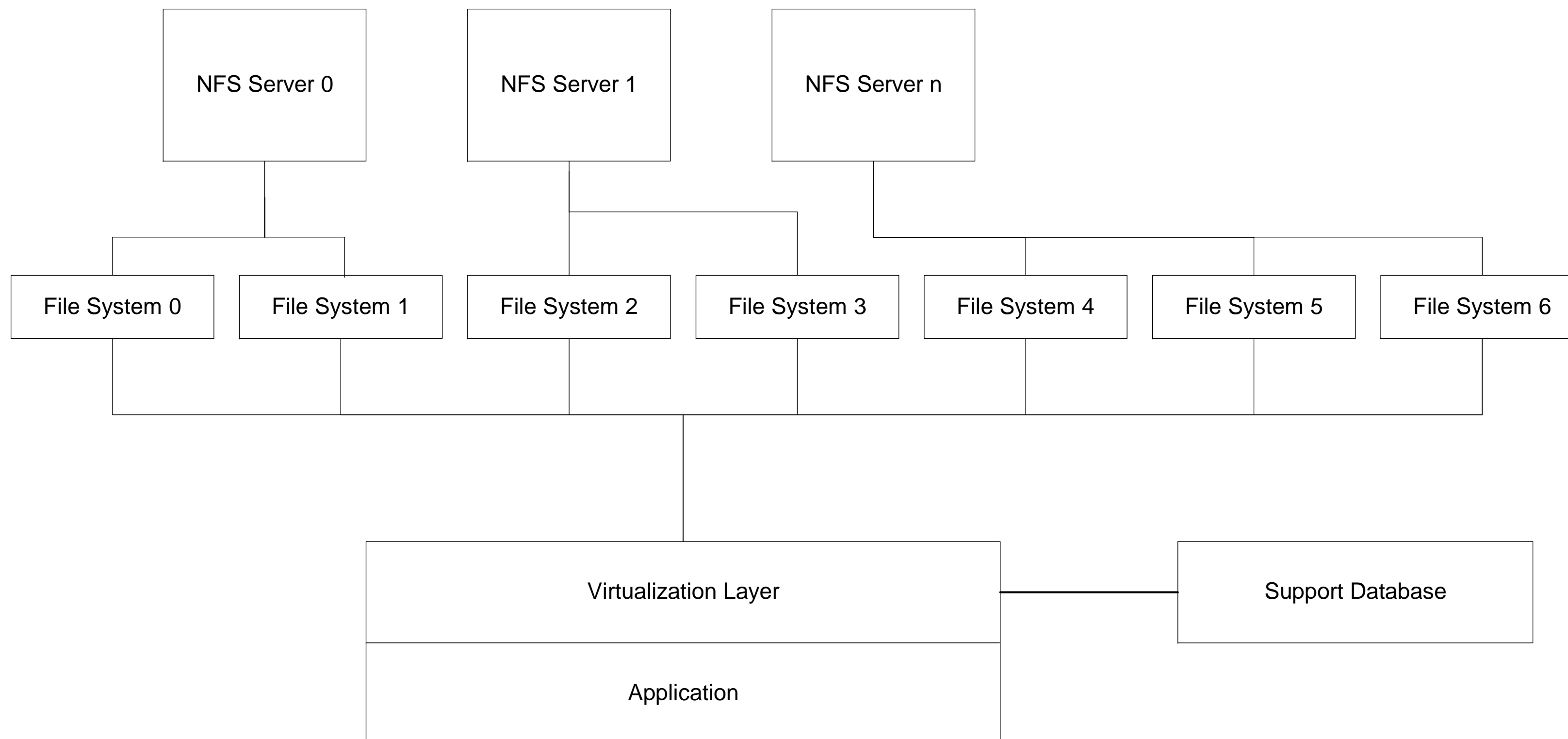
Why do “File System Virtualization” in Python?

- CFS is “application level” and runs in user space
- Version 1.0 developed with a staff of 1, so the ability to produce prototypes quickly and move concepts to production code rapidly was important
- Wanted to use build a parallel database management system quickly, so MySQL with Python seemed like an obvious choice.
- Performance was considered as a potential roadblock:
  - Can take a few seconds per transaction
  - Our application suite typically runs for hours, so “several seconds” not an issue
  - Could be an issue for real-time applications that open and close many files very quickly.

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

File System Virtualization



File System Virtualization: Mapping multiple and potentially disparate file systems into a monolithic view such that applications, scripts, etc. need not know the physical location of the files that are being manipulated.

Copyright © 2000-2005 Compete™ All Rights Reserved.

File System Virtualization

- Common directory structure for all participating file systems
- CFS uses Scheduler to select “next” device
- Database handles logical file name mapping to physical file name mapping
- “Unix-style” logical file name, for example:

```
/unique/2005-01-09_to_2005-01-15-unique-weekly.rollup
```

might map to

```
/eng8/cfsfs/unique/2005-01-09_to_2005-01-15-unique-weekly.rollup
```

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

End User Perspective

- One requirement was to integrate cleanly with Unix scripts
  - We use Unix scripts to drive our Data Pipeline application suite in PBS
  - Several applications do typical "data streaming" via pipes:

```
gunzip -cd /dir/foo.gz | myApplication | gzip - >/dir/foo-out.gz
```

- We solved this using cfsopen command:

```
cfsopen -file <fileName> --mode [r|w]

inFile=`cfsopen --file /dir/foo.gz --mode r`
outFile=`cfsopen --file /dir/foo-out.gz --mode w`
gunzip -cd ${inFile} | myApplication | gzip - > ${outFile}
```

The `cfsopen` command maps logical file name to physical file name for use in the script

Copyright © 2000-2005 Compete™ All Rights Reserved.

End User Perspective

- Another requirement was to have a programmable API
  - Compete Data Access Layer (CDAL) manages many different data sources
  - We wrapper CFS functionality behind CDAL:

```
Python 2.2.2 (#1, Oct 15 2002, 16:50:29)
[GCC 2.95.3 [FreeBSD] 20010315 (release)] on freebsd4
Type "help", "copyright", "credits" or "license" for more information.
>>> from compete.cdal import cdal
>>> data = cdal.cdal()
>>> fName = data.open( "/categories/2004-12-top-sites-monthly.rollup", "r" )
>>> fP = open( fName, "r" )
>>> line = fP.readline().strip()
>>> print line
2025427739 0       135127040       134799700       1209152000      1209151872
>>>
```

- Again, we are mapping logical file names to physical file names.

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

End User Perspective

- We wanted users to be able to catalog and delete files as needed:

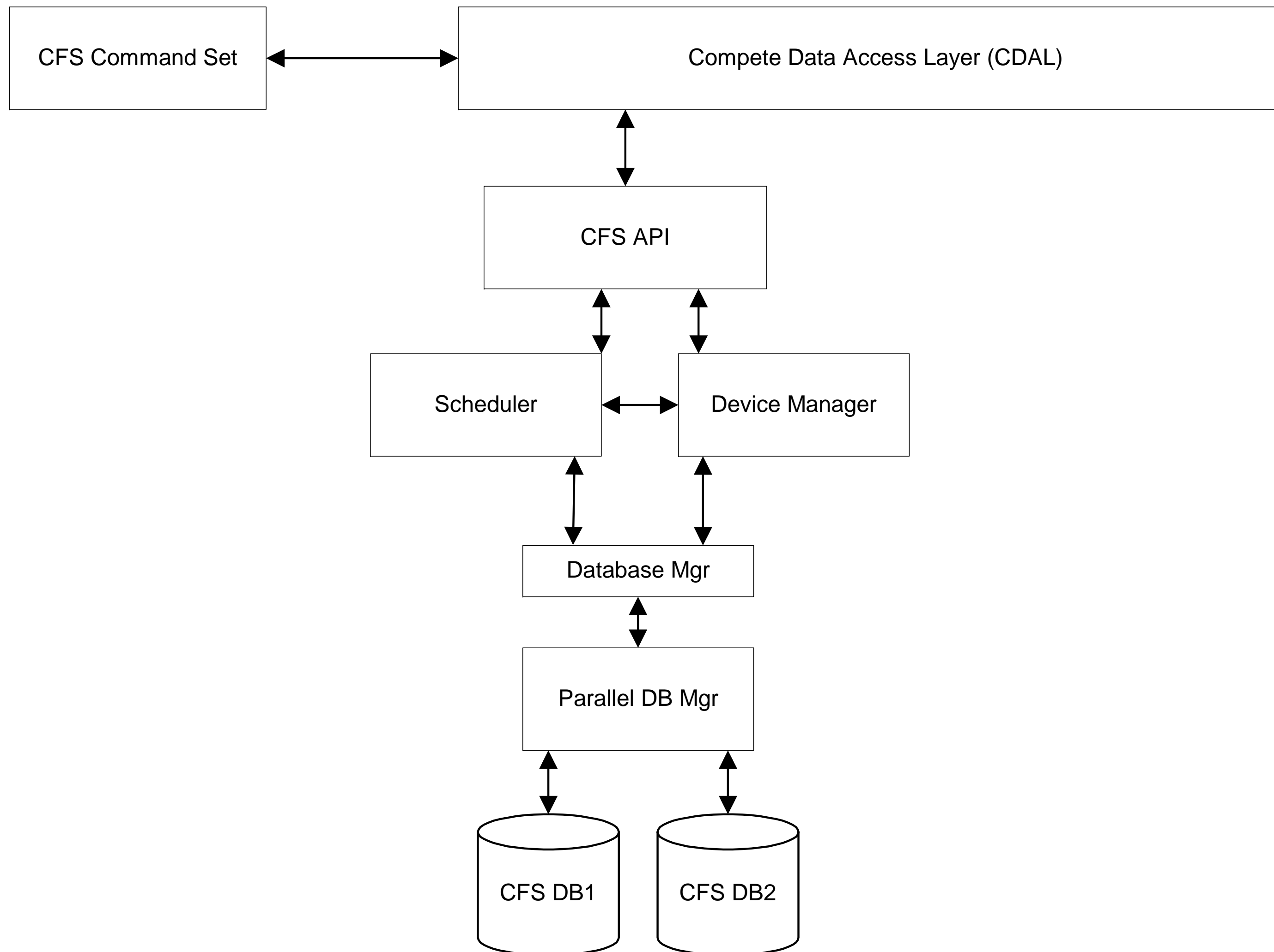
```
% cfsls --long /categories/2004-12*rollup
-rwxr-xr-x 1 compete dev 243127243 Dec 13 09:11 /categories/2004-12-05_to_2004-
   12-11-categories-weekly.rollup
-rwxr-xr-x 1 compete dev 233573907 Dec 20 16:59 /categories/2004-12-12_to_2004-
   12-18-categories-weekly.rollup
-rwxr-xr-x 1 compete dev 211203555 Dec 27 03:44 /categories/2004-12-19_to_2004-
   12-25-categories-weekly.rollup
-rwxr-xr-x 1 compete dev 226610879 Jan 08 08:04 /categories/2004-12-26_to_2005-
   01-01-categories-weekly.rollup
-rwxr-xr-x 1 compete dev 2508 Feb 11 12:43 /categories/2004-12-top-sites-
   monthly.rollup
-rwxr-xr-x 1 compete dev 562855514 Jan 05 10:46 /categories/2004-12_categories-
   month.rollup
```

```
% cfsrm --file /unique/2004-10_unique.rollup
```

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

Compete File System Architecture



Copyright © 2000-2005 Compete™ All Rights Reserved.

Architectural Issues in Version 1

- Wanted to build a "straightforward" implementation running user space
  - Map logical file names to physical file names
  - Allow access to CFS files as "normally" as possible from command line & programs
  - A "stateless" model made a lot of sense:
    - No daemons to worry about
    - Increased likelihood that CFS can port to other platforms
    - Straightforward code easy for others to enhance and maintain

- CFS maps logical file names to physical file names
  - Database used to perform the mapping

- CFS uses a scheduler to assign file systems to use during write operations
  - Database used to contain information used by the scheduler

- So the database is very important
  - What if it FAILS?
  - Parallel database implementation

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

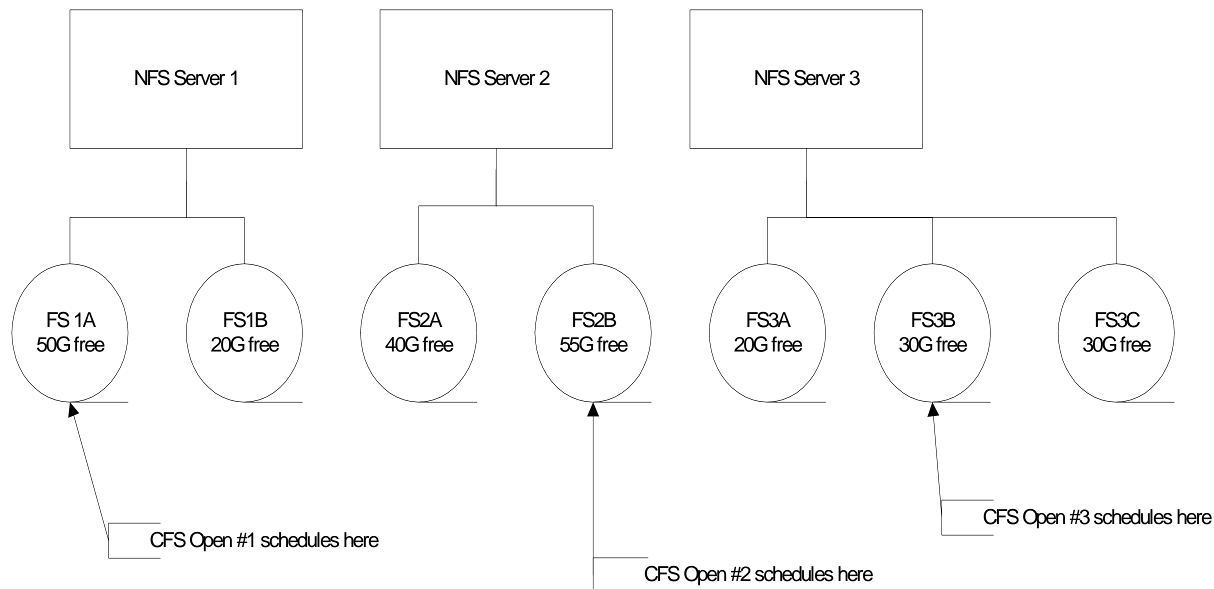
Scheduling and Load Balancing

- **Multiple file systems exist on individuals NFS servers**
- **In version 1, we presumed all file systems were essentially the same size**
- **Used a “round robin” approach to scheduling:**
  - Listed devices in a table alternating according to NFS server
  - Use a database table to determine the “next” device in the chain on opens-for-write
- **Round robin worked fairly well…**
  - Loads fairly even on the NFS servers
  - Began to see issues with file system capacities
    - Some files in CFS were very large, others very small

Copyright © 2000-2005 Compete™ All Rights Reserved.

The Best-Fit Scheduler introduced in Version 1.1

- Best Fit Scheduler tries to select the best file system based on size
- Round-robin iteration across NFS servers
  - Select largest file system available on that server
  - Introduced device thresholds that eliminate file systems from consideration when they get small



COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

Best Fit Scheduler results

- Load balancing remains equally as effective

- This case causes problems:

```
filePointers = []
for fileNumber in range(5000):
    fileName = "/osvData/2004-12-01_to_2004-12-31_"+str(fileNumber)+".dat"
    fP = data.open( fileName, "w" )
    filePointers.append( fP )
```

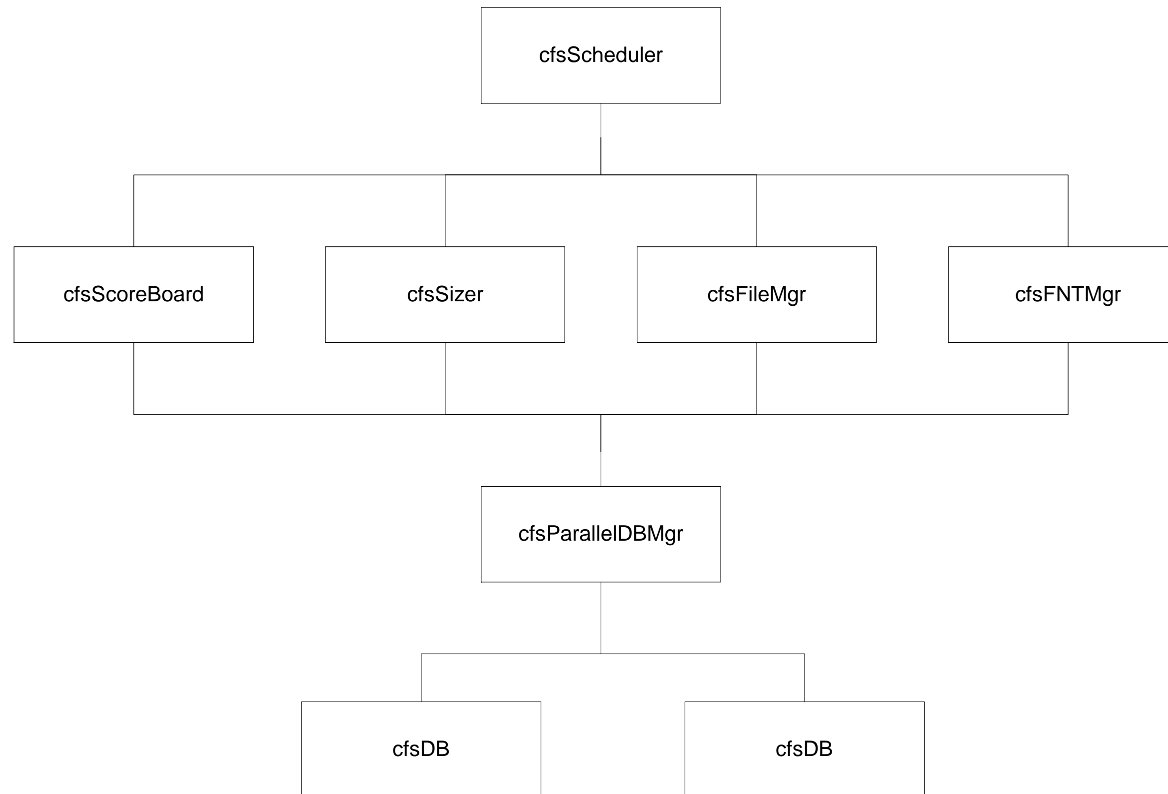
- In this case, all the files get allocated on a few file systems – if the files are large then the file system can fill up – not a good thing. Round Robin tended to avoid this by distributing files evenly across the file systems.

- This is not a made up case – one of our applications writes out many many temp files (or think of the “sort” command in Unix)

Copyright © 2000-2005 Compete™ All Rights Reserved.

Scheduler Work in Progress

- Predictive Scheduler:
  - Track file system utilization by score-boarding file system sizing
  - Predicts file sizes of new files in the same directory by tracking average sizes
  - Dynamically tracks implicit file closes by monitoring process group IDs
    - This "file open reaper" runs periodically out of cron – could be a daemon if we chose



COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

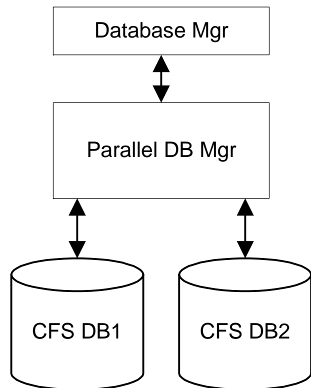
CFS Internal State Modeling and Failure Prevention

- Running instances of CFS are stateless for ease of implementation
- State information is cached in a database
  - Logical to Physical File Mappings
  - Open file information in v2.0
  - Device selection and eligibility information
  - Etc.
- Losing the CFS database due to some failure would be catastrophic
- CFS uses parallel databases
  - Two separate physical machines
  - Two separate disk arrays on non-CFS UFS storage RAID-5
  - API issues queries under locking on both databases
  - Journaling of queries allows replay to recover from a failure

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

CFS Internal State Modeling: Parallel Databases




```
def deleteFromCache(self, logFileName):
        try:
            if self.hasDeleteRights():
                if self.__lock(self.__cacheTable, "write"):
                    query = makeQuery( “delete”, logFileName)
                    self.__db[ PRIMARY_DB ].execute( query )
                    self.__db[ SECONDARY_DB ].execute( query )
                    self.__updateLogs( query )
                    self.__unlock()
                    return 0
                else:
                    return 1
            else:
                return 1
        except:
            self.__unlock()
            raise
```

COMPETE, INC. The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

Role of Python in Building CFS

- **Robust class libraries and modules made it easy to think about algorithms, functionality, without worrying about whether a particular bit of functionality was available or supported.**
- **Consideration given to other languages for deployment - C, C++ (Java considered evil by mgmt), but rejected for the "usual" reasons:**
  - Rapid deployment easy with Python, not necessarily so with other languages
  - Class library and 3rd party module support especially for MySQL
  - Object-oriented nature of Python made it easy for developers to work on code without stepping on each other
  - Can we please stop calling it a “scripting language”?
  - Personal perspective: Coding in Python is “more fun per hour” than other languages

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

Whining and Ranting

- **Python VM footprint and speed are good - but a compiled language would be better**
  - Python VM footprint and speed are good - but a compiled language would be better

- **A better interface to foreign code would have allowed us to do more in more efficient ways**

- **What is up with I/O in Python?**
  - Not possible to write code to deal with hundreds of gigabytes of data
  - Even SWIGging in high performance file manipulation code runs orders of magnitude slower
  - Bottom line is that most CFS usage is from the command set and drives C/C++ code

COMPETE, INC.
The Predictive Edge

Copyright © 2000-2005 Compete™ All Rights Reserved.

So is Python Bad?

- Emphatically no - it does surprisingly well in this very "not a script" application
- Ability to develop massive functionality quickly, and to debug and test easily allows a very small organization to do big work fast
- We don't think in terms of language-based religious wars: this stuff just works well for us so we use it
- Bottom Line: I have budget, time, and headcount to implement CFS however I want, and we are still using Python

Copyright © 2000-2005 Compete™ All Rights Reserved.

The Final Analysis: Success of CFS

Results of Building and Deploying CFS

- Multiple terabytes of data under coherent structure
- Hundreds of thousands of files across multiple devices
- Effective storage resource management
- Zero worries
- Happy Management – the hallmark of good software development ☺

Copyright © 2000-2005 Compete™ All Rights Reserved.