IPDPS 2013, 5/21/2013

Efficient and Scalable Retrieval Techniques for Global File Properties

Dong H. Ahn, Bronis R. de Supinski, Todd Gamblin, Gregory L. Lee, Matthew P. LeGendre, Adam Moody, and Martin Schulz
Lawrence Livermore National Laboratory

Michael J. Brim and Barton P. Miller
University of Wisconsin

Lawrence Livermore National Laboratory

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

LLNL-PRES-636652

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. It was supported in part by Department of Energy grants DE-SC0004061, 07ER25800, DE-SC0003922, DESC0002153, DE-SC0002154, and DE-SC0002155.

Efficient file accesses are becoming increasingly important and challenging

- Large-scale system sizes continue to grow
- This exponential growth in concurrency makes efficient file accesses increasingly important
- But optimizing file accesses require detailed run-time knowledge of file systems and location(s) of files on them
- HPC does not have a common, scalable way to retrieve such global file information

Program start-up manifested as a denial-of-service attack: A lesson learned

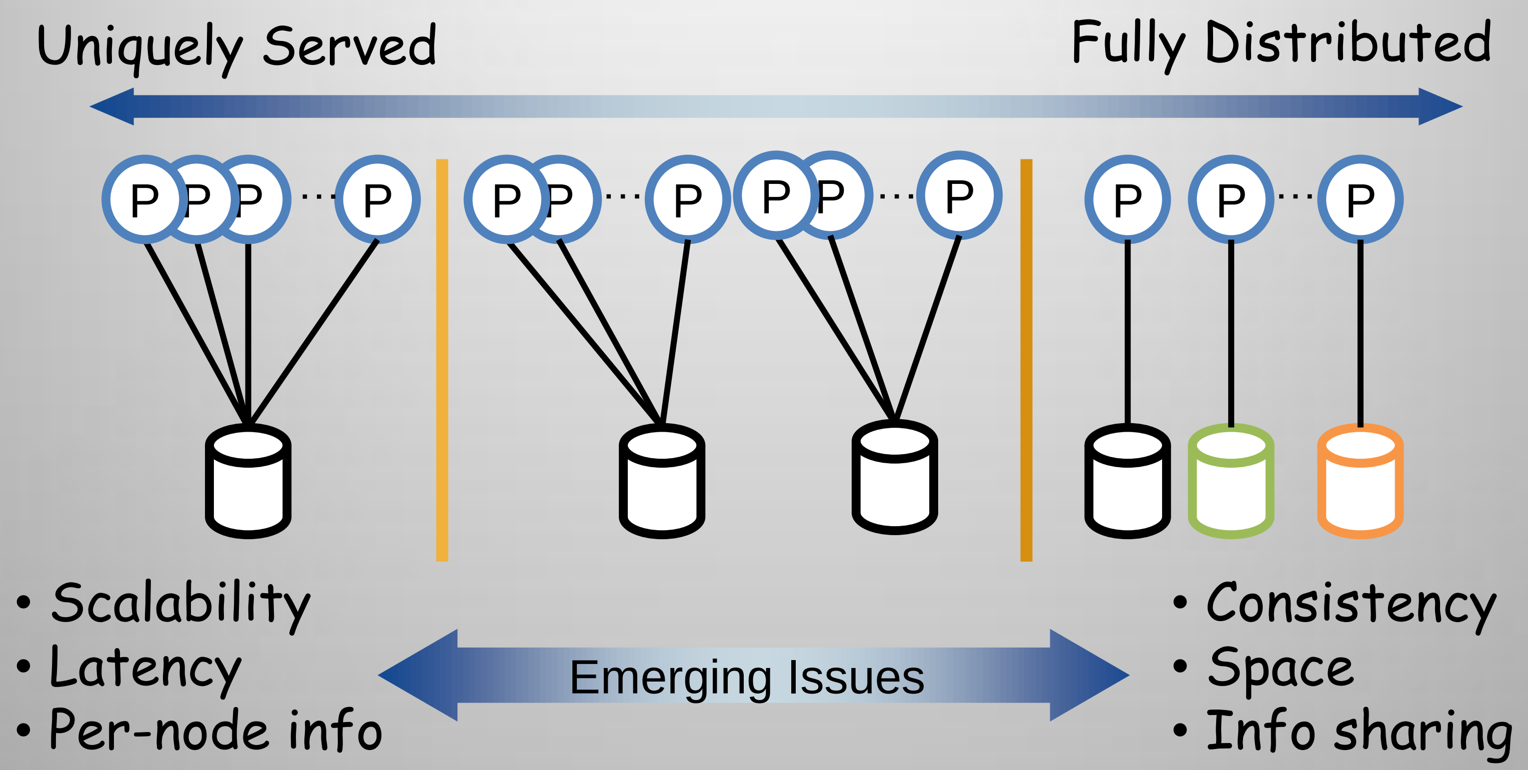
- KULL: a large, mission-critical multi-physics simulation code
- When this application was first run on DAWN, program start-up appeared to scale very poorly
- Start-up significantly disrupted the entire computing facility
- 16,384 instances of the dynamic loader (ld.so) were making combined 300 million open calls to an NFS server
  - 16K X 20 (lib search paths) X ~1000 (dependent shared libs) = 300M !

All software elements on extreme-scale machines must efficiently use file systems

- Challenges go beyond large dataset access patterns: dynamic loader, run-time tools, input-deck readers, scripting languages etc
- Must optimize their file access schemes and consider a trade-off between communication and file accesses
- Optimization requires detailed run-time information of file systems and location(s) of files on them
- Non-trivial: today's machines mount many file systems with different performance characteristics

Need scalable, general-purpose mechanisms and abstractions to retrieve global file properties: Fast Global File Status (FGFS)

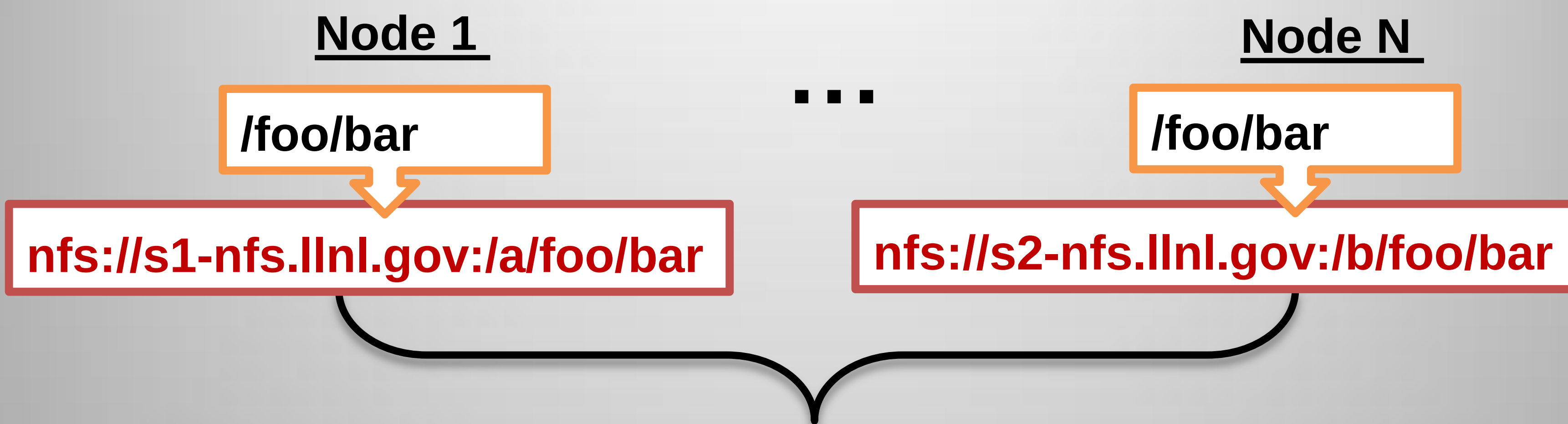
The trade-off space: HPC file distribution models introduce many different issues



FGFS is a query layer that assists HPC software in making I/O trade-off decisions

- Responsible for **scalably classifying files and file systems**
- Supports I/O trade-off decisions for a wide range of HPC software
- Directly with the software itself or through a global file I/O coordinator

Key idea for scalability: extracting global properties through name comparisons



- Is this file uniquely served?
- Is this file fully distributed?
- Will N simultaneous accesses thrash file systems or not?
- …

MountPointAttributes resolves a local file path into URI with no file-system access

```
string & resolvePath(const char *pth) {
  string uriStr;
  FileUriInfo uriInfo;

  MountPointInfo mpInfo(true);
  mpInfo.getFileUriInfo(pth, uriInfo);
  uriInfo.getUri(uriStr);

  return uriStr;
}
```

node1

```
void manageConfigs() {
  char *lid1="/etc/tool/conf";
  char *lid2="/usr/etc/tool/conf";
  char *lid3="/home/joe/.tool/conf";
  char *lid4="/lscracta/j_cwd/conf";

  string gid1 = resolvePath(lid1);
  string gid2 = resolvePath(lid2);
  string gid3 = resolvePath(lid3);
  string gid4 = resolvePath(lid4);
  ...
}
```

file://node1/etc/tool/conf

nfs://s1-nfs.llnl.gov:/e/usr/etc/tool/conf

nfs://dip-nfs.llnl.gov:/v/joe/.tool/conf

lustre://172.16.60.200:/tmp/j_cwd/conf

Global File Status queries capture our HPC file distribution models and pertaining issues

- The global namespace forms a reference space where parallel name comparisons extract global properties
  - the number of different sources
  - the process count and the representative process of each source
- ***FgfsParDesc*** is a primitive that returns this info
- ***GlobalFileStatusAPI*** exposes the HPC file distribution models
  - ***isUnique(), isFullyDistributed()***
  - ***isWellDistributed(), isPoorlyDistributed()***
  - ***isConsistent()***
- Support for both synchronous and asynchronous I/O patterns
  - ***SyncGlobalFileStatus***
  - ***AsyncGlobalFileStatus***

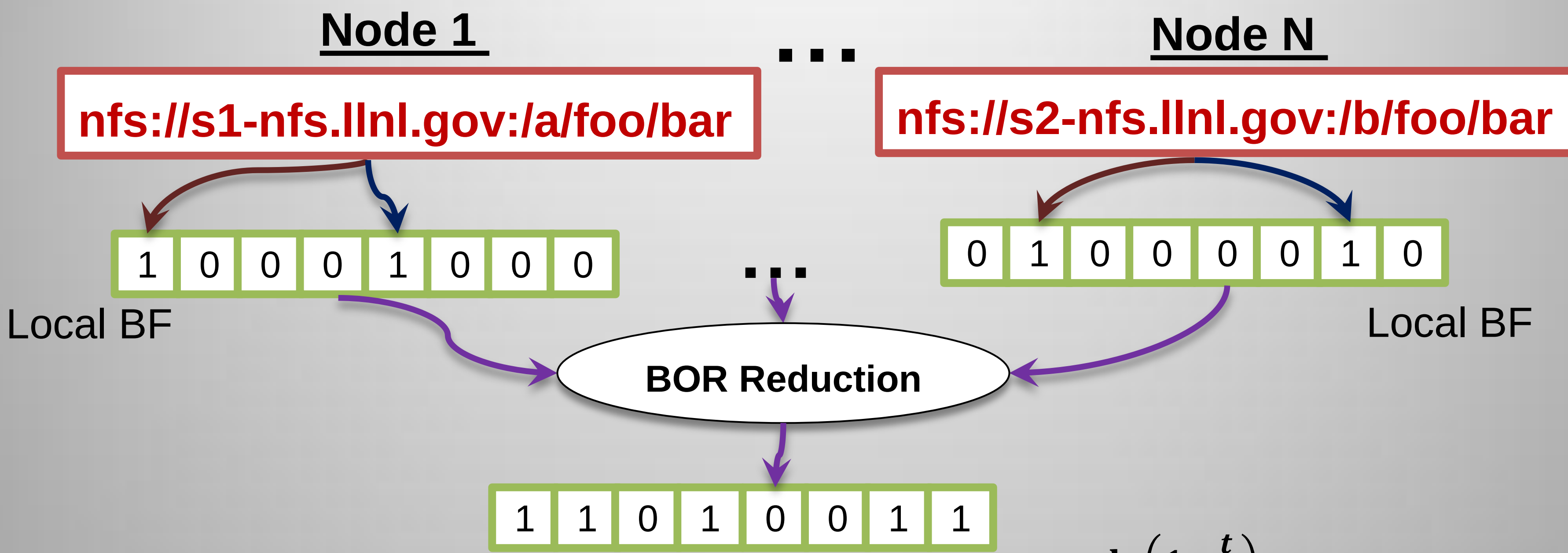
A highly scalable reduction algorithm extracts the degree of file distribution or replication

$$\text{file} \xrightarrow{\textbf{Raise}} \{URI(file)_0, URI(file)_1, \ldots, URI(file)_{n-1}\}$$

$$\xrightarrow{\textbf{Reduce}} \{UniqueURI(file)_0, \ldots, UniqueURI(file)_m\}$$

- Cardinality/Group-info of the reduced list conveys a global structure
- A representative of each unique source helps minimize file accesses
- A tree-based parallel reduce for the general case
  - But scales like concatenation with too many unique names
- A multilevel triaging scheme imposes a scalability bound
  - First level: a fix-sized boolean reduce to determine *isFullyDistributed()*

Next refinement: Bloom-filter-based cardinality estimation



- Maximum likelihood cardinality estimation: $\frac{\ln\left(1-\frac{t}{m}\right)}{k * \ln\left(1-\frac{1}{m}\right)}$
  - m num of bits, t is the num of true bits, and k is the num of hash functions
- Set the Bloom-filter density to be 50% with respect to the worse case
  - The worst case for billion-core machine needs ~150KB

Global file systems status queries retrieve file systems that meet global properties requirements

- Inverse function of global file status queries
  - Given a set of required global properties of a file system, what are the best matching locations?
- ***GlobalFileSystemsStatus***
  - Is passed a ***FileSystemCriteria*** object
  - Mandatory ***space*** requirement, and optional ***speed***, ***distribution***, and ***scalability*** requirements
- A scoring function estimates performance and orders *qualified* file systems

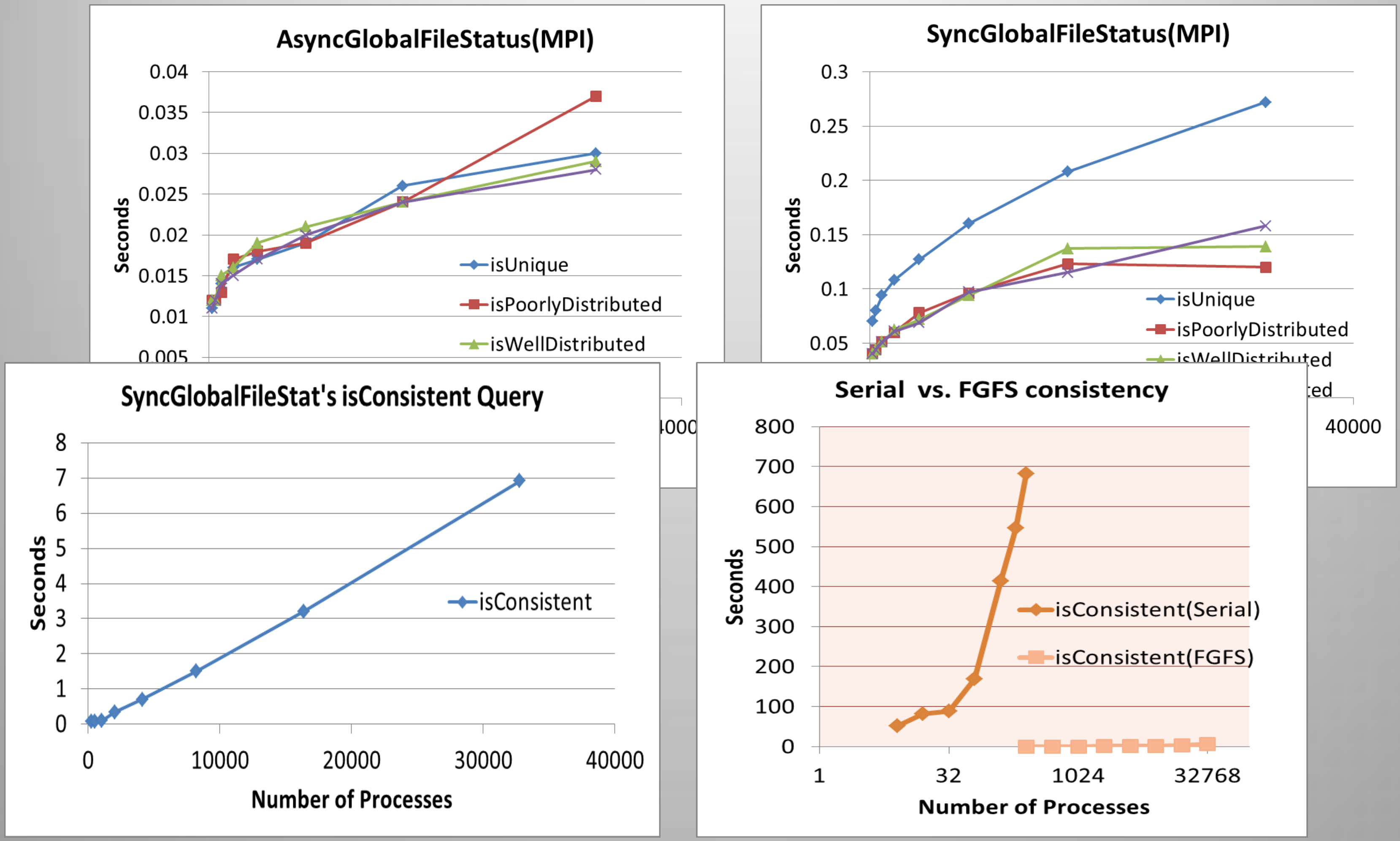
$$\frac{\boldsymbol{Scalability(file\ system)}}{\mathbf{M}\boldsymbol{ax(Scalability(file\ system), Distribution(file\ system))}} * \boldsymbol{Speed(file\ system)}$$

LLNL-PRES-636652

Our experiments are to evaluate FGFS' capability of assisting file access optimization

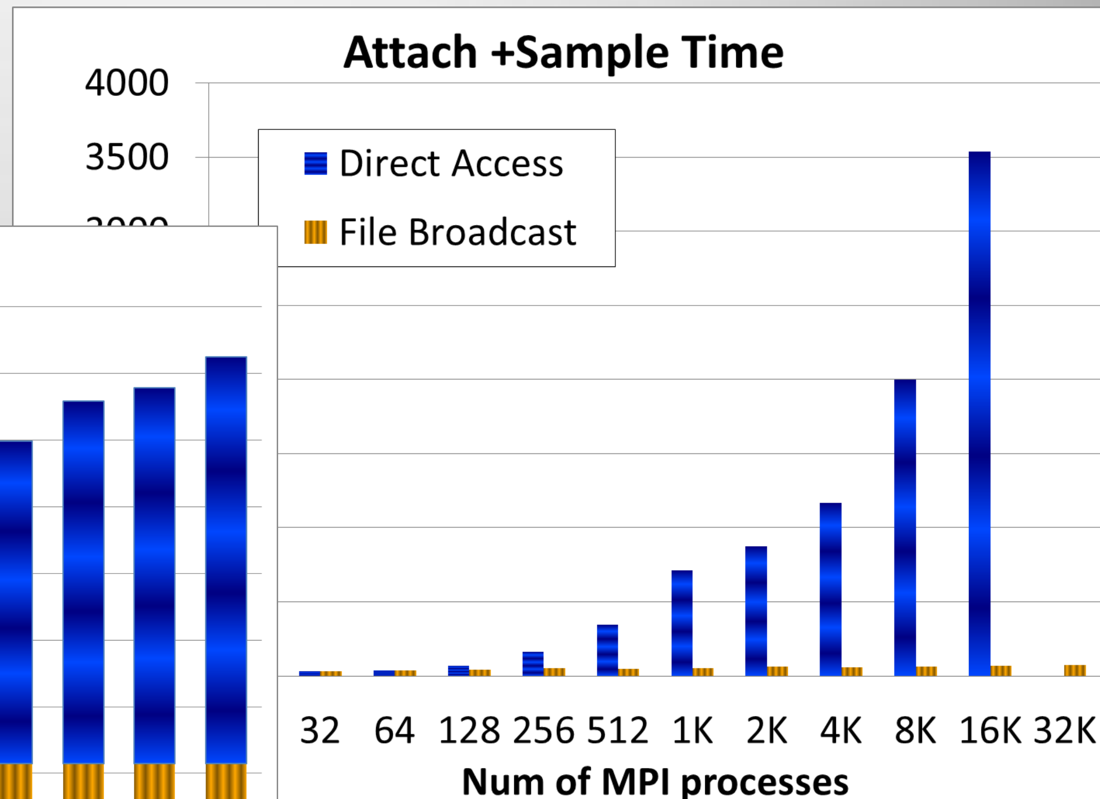
- Primary evaluation goals
  - The performance and scalability of various FGFS queries
  - The effectiveness and utility of FGFS on a variety of HPC software
- Controlled experiments and three case studies
  - Benchmark FGFS performance on three multi-physics applications
  - Integrate FGFS to HPC elements with vastly different characteristics
- Ran on Linux clusters installed at LLNL
  - 2-socket x 8-core Intel Sandy Bridge (2.6GHz) with 32 GB of RAM
  - The largest cluster (Zin) with up to 2,916 compute nodes = 46,656 cores
  - Qlogic Infiniband QDR interconnect

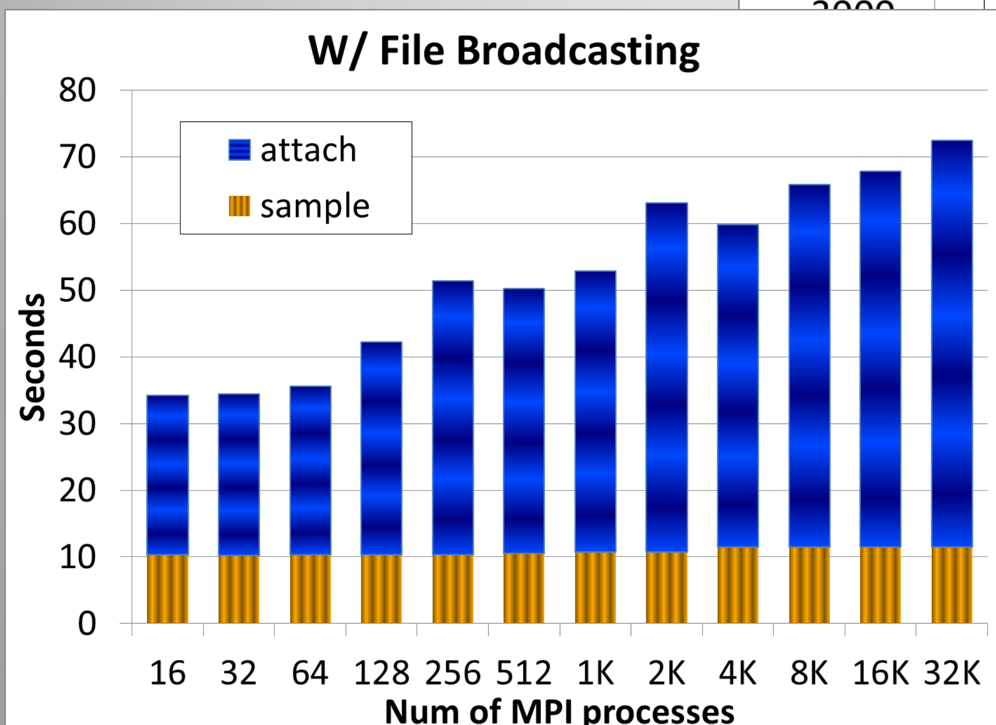
Most file status queries on KULL (w/ 848 shared libraries) complete in 272 msecs at 32K procs



FGFS addressed a scalability challenge in STAT's accessing of file systems

Log scaling with $R^2 = .958$





ULL (with big executable mode)

FGFS serves as a key component of a novel massive parallel loading service

- SPINDLE (Scalable Parallel Input Network for Dynamic Loading Environment)
- SPINDLE file-cache servers form a tree-based network and coordinate file-system accesses of the dynamic loader.
- SPINDLE servers use ***AsyncGlobalFileStatus*** to choose between a direct file-system access and file broadcasting.
- The Pynamic benchmark was shown to scale well up to 15,360 MPI processes with no disruption to shared file systems

We will present details of SPINDLE at ICS (6/10/13 – 6/14/13, Eugene, Oregon).

FGFS facilitates efficient, non-disruptive use of file systems for a wide range of HPC software

- Efficient files accesses are increasingly important and challenging
- Developed Fast Global File Status as a scalable, portable mechanism to retrieve global information on files or file systems
- FGFS queries are highly scalable and provide orders-of-magnitude improvements over traditional approaches
- Various case studies suggest that FGFS can be effective for a wide range of HPC software elements
- FGFS will deeply be integrated into various HPC software systems, extending its benefits to many essential elements of HPC

- MountPointAttributes has been released: http://dongahn.github.io/MountPointAttributes
- Other components coming soon.

Lawrence Livermore National Laboratory