UNIVERSITY OF NOTRE DAME

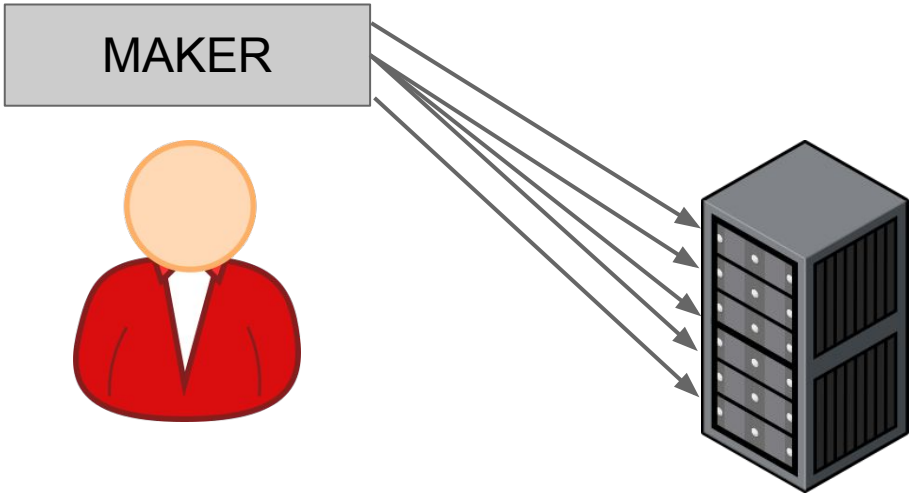
# Streaming Community Detection for Partitioning Parallel Filesystems

CCTools

Tim Shaffer

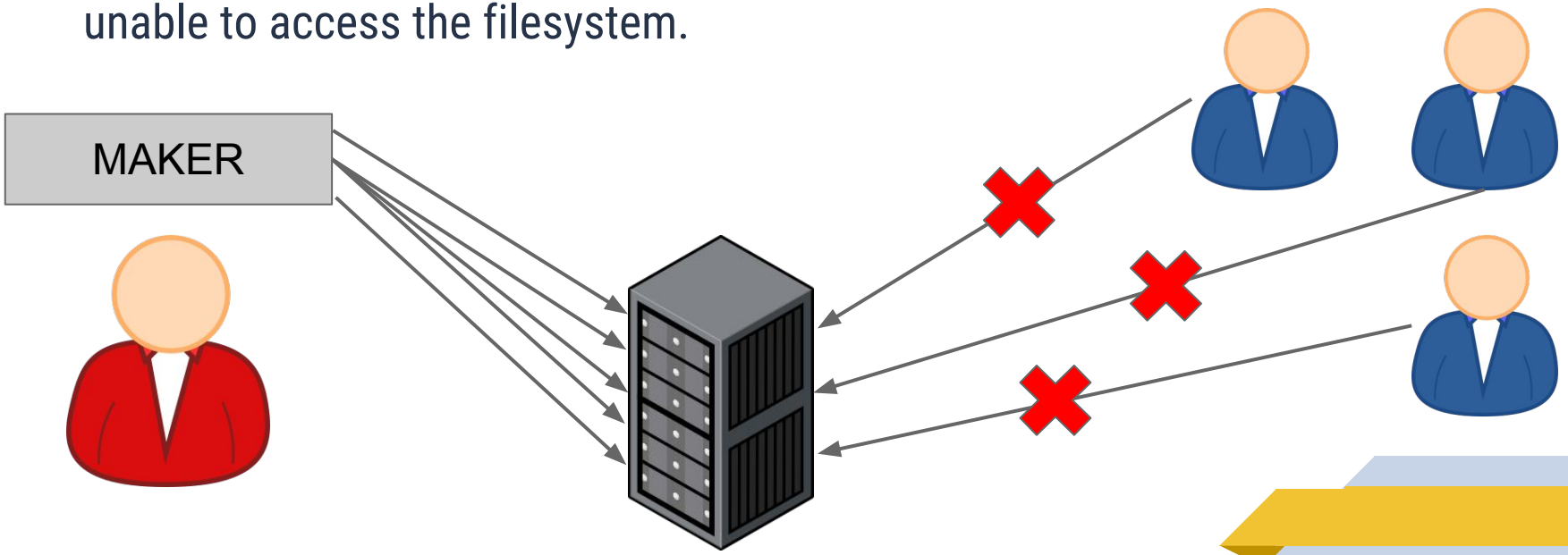
# Motivation

A (well-meaning) user tried to run a bioinformatics pipeline to analyze a batch of genomic data.



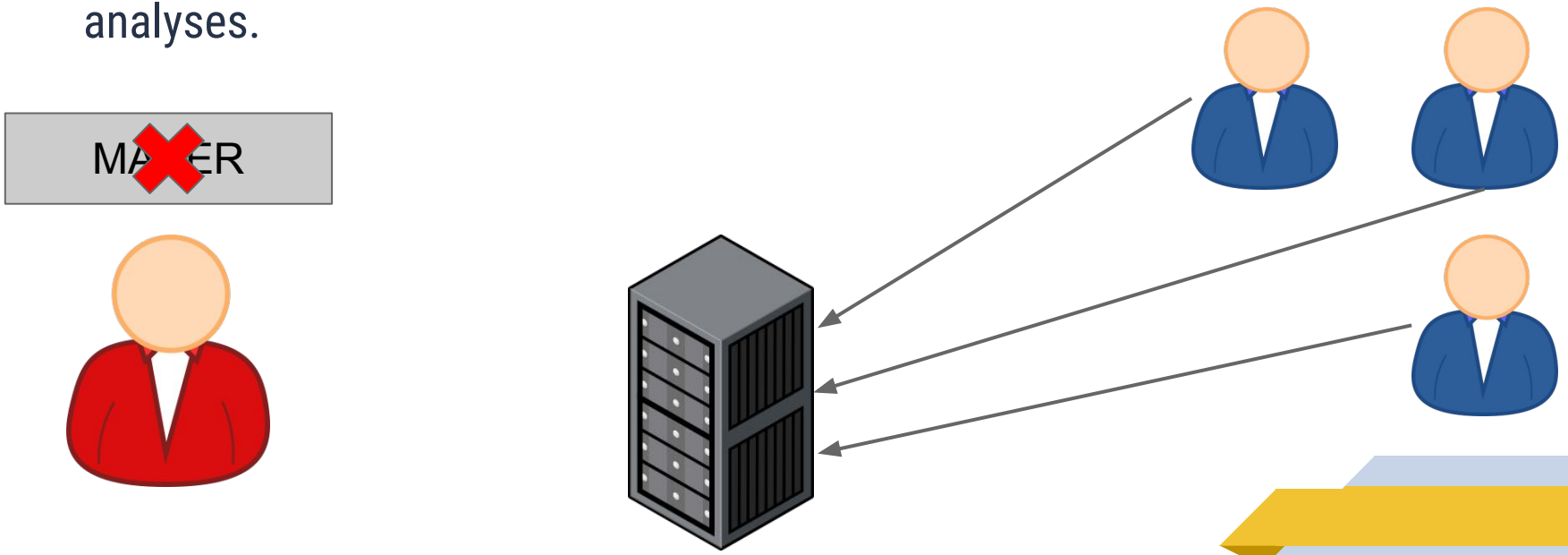
# Motivation

Shared filesystem performance became degraded, with other users unable to access the filesystem.



# Motivation

That user got a strongly worded email and had to stop their analyses.



# Metadata Storm

Certain program behaviors produce **large bursts** of metadata I/O activity (e.g. library search).

These behaviors can occur at the **same time across multiple workers** (e.g. startup, new analysis phase).

With a large number of nodes, the timing and intensity of metadata activity align to **overwhelm the shared FS**.

# Existing Approaches/Related Work

Shared filesystems can scale up their metadata capacity.

Panasas, Ceph, etc. use multiple metadata servers to better distribute the load.

General purpose solution

# Existing Approaches/Related Work

Applications can use a metadata service layered on top of the shared filesystem (e.g. BatchFS, IndexFS).

More efficient metadata management than the native filesystem

Allows for client-side caching and batch updates

# Case Study: CVMFS

Software used in analyzing LHC data is distributed through CVMFS

Includes multiple layers of caching and load balancing to handle bursts of activity.

https://cernvm.cern.ch/portal/filesystem

# Case Study: CVMFS

CVMFS can be difficult to deploy at some sites.

- Requires all workers to have fast internet access
- Requires root access on workers to set up FUSE module
- Large total size (terabytes), though each worker only uses a small part
- Each project defines its own filesystem layout for software packages/frameworks

# Idea: Manually transfer CVMFS data

For network issues, populate local Squid cache with required data

For root/FUSE limitations, build a static image (Docker, Singularity, etc.) containing required pieces

But we don’t actually know what’s required...

# Profiling CVMFS Applications

Analysis is data-dependent and non-deterministic, so we need to profile a large number (thousands) of analysis runs

Each run makes a large number of filesystem accesses (millions)

We need to use `strace`-type events to infer filesystem organization.

# Profiling CVMFS Applications

```
18212 1503501245.079960 read(3</lib64/libpthread-2.12.so>,
"\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x03\x00\x3e\x00\x01\x00\x00\x00"..., 832) = 832
```

# Constructing a Graph from Execution Profiles

Simplifying assumptions:

- Each analysis run is an independent serial process.
- Filesystem accesses follow the Markov property

Turns out to hold in many cases (e.g. sequence of operations for loading libraries or `$PATH` search)

We can easily ignore data access and focus on metadata

# Constructing a Graph from Execution Profiles

In this graph, nodes are **filesystem entries (inodes)**.

Edge weights indicate the **number of times** the access pattern

*inode A -> inode B*

occurred over all runs.

Amenable to streaming updates

## Community Detection

Groups of filesystem entries frequently accessed together are visible as communities in the execution graph.

**Hierarchical community detection** allows us to identify good shards/partitions for manual distribution.

Streaming algorithm exists

# Sequential algorithm: Girvan–Newman

Progressively removes edges from graph

The remaining components are the communities.

Uses **edge betweenness**: the number of shortest paths between pairs of nodes that run along an edge

# Sequential algorithm: Girvan–Newman

Pseudocode of algorithm (courtesy of Wikipedia)

1. For each edge *E* in *G*, compute the betweenness of *E*.
2. Remove the edge with highest betweenness from *G*.
3. Recalculate betweenness for edges affected by the removal.
4. Repeat Steps 2 and 3 until no edges remain.

Results in a dendrogram showing successively finer clusters

# Improvements

Computing edge centrality is expensive, must be (partially) computed after each edge removal.

Sequential algorithm (Girvan–Newman) runs in $O(VE^2)$ or $O(V^3)$ in a sparse graph.

STINGER supports streaming updates and parallel agglomerative clustering.

# References


https://cernvm.cern.ch/portal/filesystem

https://en.wikipedia.org/wiki/Girvan%E2%80%93Newman_algorithm

http://www.stingergraph.com/

M. Girvan and M. E. J. Newman, Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002).

M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks. Preprint cond-mat/0308217 (2003).