UNIVERSITY OF NOTRE DAME

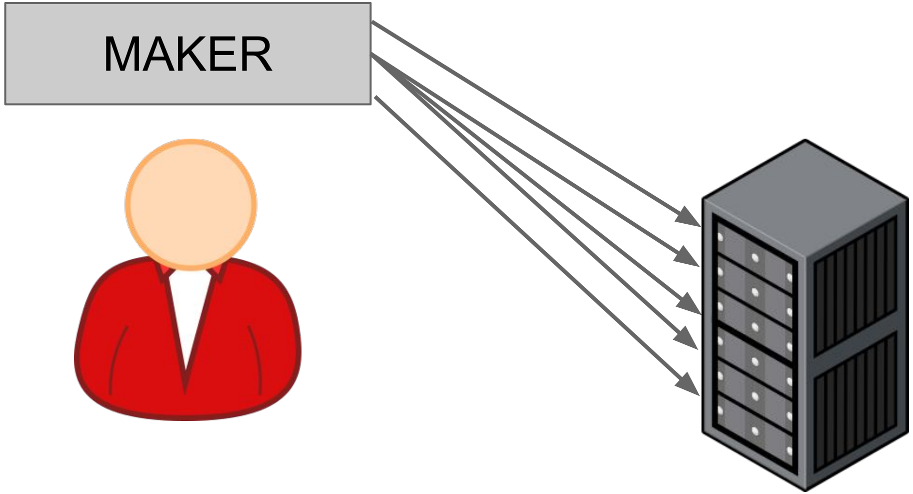
Streaming Community Detection for Partitioning Parallel Filesystems

Tim Shaffer

CCTools

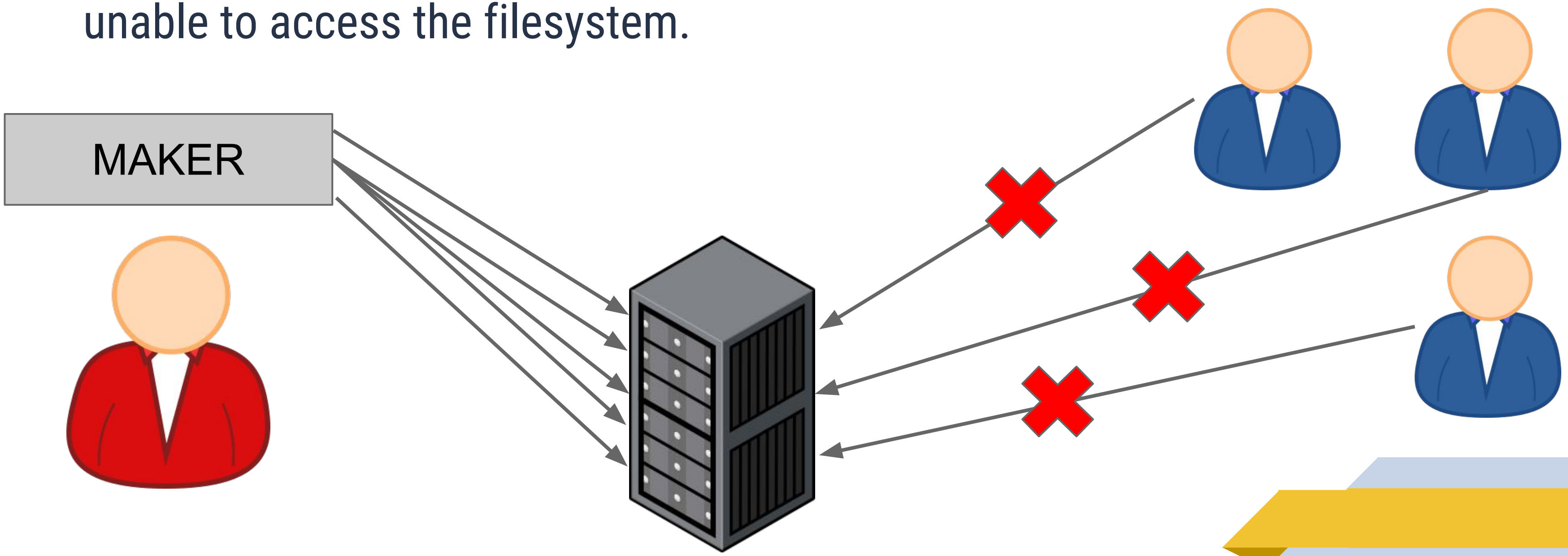
Motivation

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



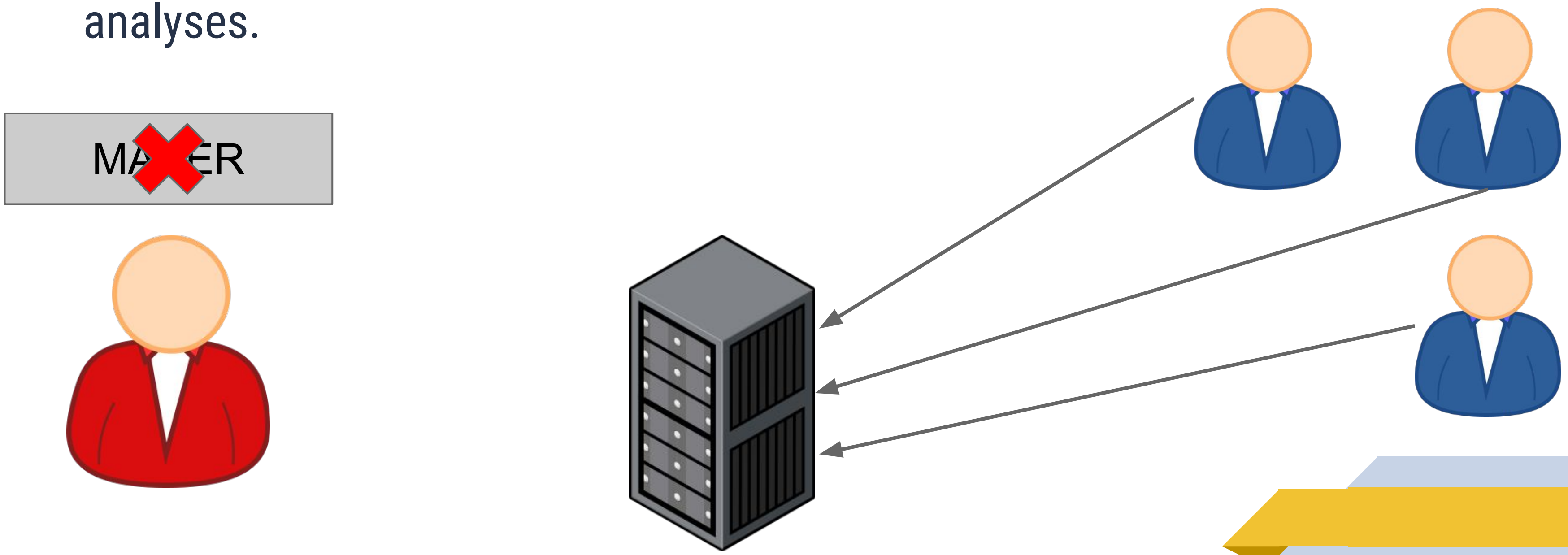
Motivation

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).

Techniques such as worker-side caching and pre-staging can help, but how do we know which parts of the FS would benefit?

Profiling Scientific Applications

Analysis pipelines have many components, show data-dependent and non-deterministic behavior.

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

We need to use `strace`-type events of numerous analyses to profile filesystem interactions.

Profiling Scientific Applications

```
29204 open("/etc/passwd", O_RDONLY|O_CLOEXEC) = 3
```

Constructing a Graph from Execution Profiles

CCTools

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 for this

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

Complexity

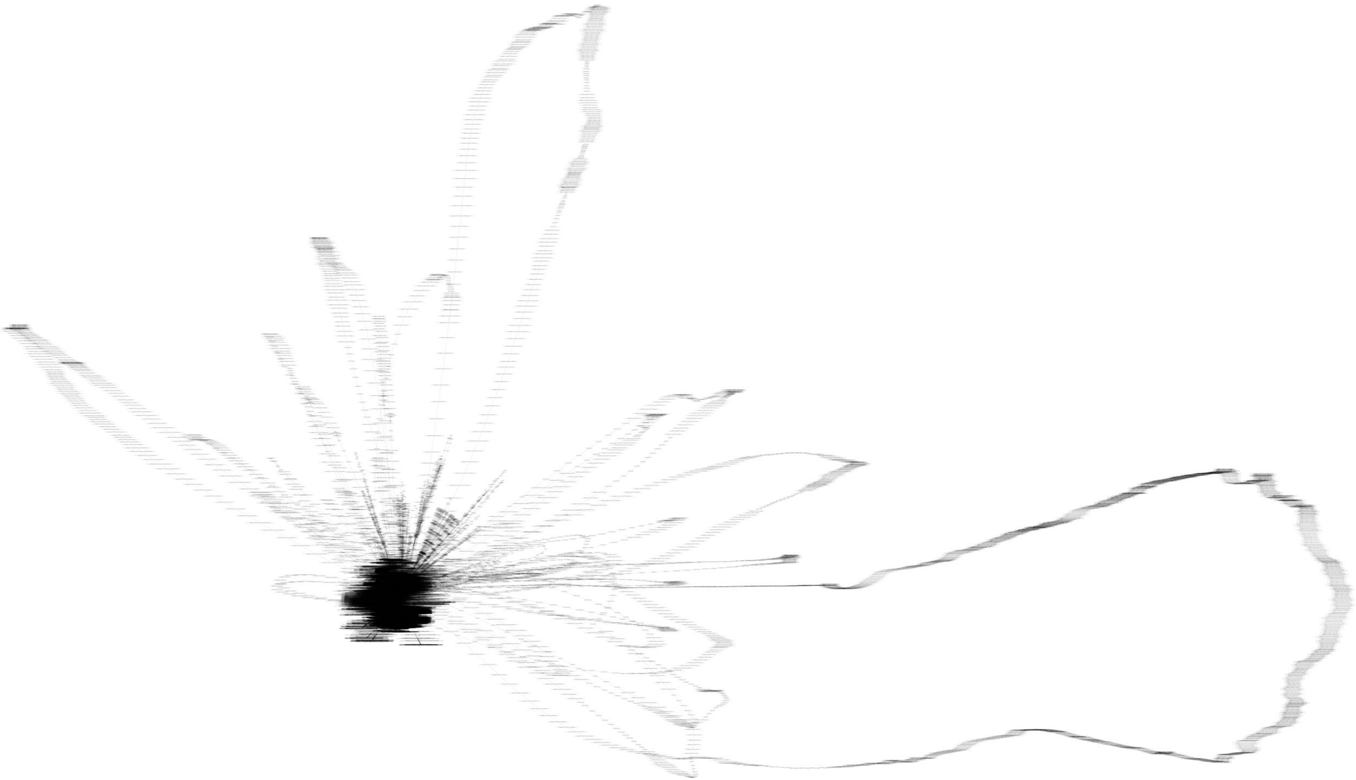
Computing edge centrality is expensive, must be (partially) computed after each edge removal.

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.

Data Sets

	Events	Nodes	Edges
`true`	47	46	45
`bash`	5,499	840	1,569
MAKER	1,813,544	24897	129,153



(Poorly done) Visualization of MAKER Graph

(Poorly done) Visualization of MAKER Graph

Initial Implementation

NetworkX includes an implementation of Girvan–Newman!

Straightforward Python implementation parses the `strace` logs, constructs the graph, and invokes Girvan–Newman.

Dendrogram (arbitrarily) limited to k=5 levels deep.

Performance Results

	Girvan–Newman	1st Betweenness Centrality
`true`	0.43 s	0.41 s
`bash`	290 s	10. s
MAKER	😭	?? (> 45 min)

Results are (roughly) consistent with $O(VE^2)$ running time.

Enhanced Implementation

STINGER allows for streaming and parallel operations.

C rather than Python.

Heuristics to reduce the size of the graph might help.

References

http://www.yandell-lab.org/software/maker.html

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).