# Home

FileMap is a file-based map-reduce system for data-parallel computation.

## Why Map-Reduce?

The map-reduce method of parallel computing was introduced by Google and further popularized by open source implementations like Hadoop, Disco, and others. Map-Reduce is remarkable in its simplicity and scalability. Traditional parallel environments have been based either on explicit message-passing APIs or on the appearance of a global shared-memory system. In contrast, Map-Reduce provides a rigid data-flow model in which the user need only write discrete kernel functions that fit within that dataflow. This restrictive model does not support many communication patterns in parallel codes. However, it is sufficient for a large number of data-intensive computing tasks. In return for accepting these restrictions, programmers can write simple, serial functions that a map-reduce run-time can parallelize very effectively.

### What is unique about FileMap among other Map-Reduce frameworks?

FileMap is developed around several defining themes:

- File-based, rather than tuple-based processing.
  - Rather than having to provide routines that parse input data in to (key,value) pairs, your code can operate directly on input data files. If you want the first stage to be a canonicalization of data to (key,value) strings, you can still do that, but it's not required
  - Binary files (like ''pcap'' files) need not be expensively canonicalized into a string representation before processing
- Language & tool-chain agnostic.
  - Reuse existing domain-specific and POSIX file processing tools.
  - Processing elements are executable programs rather than Java classes like Hadoop. This means you can use existing tools like ''grep'', ''awk'', and "tcpdump" in your processing.
- Caching of intermediate results
  - Allows users to iteratively refine queries while agressively reusing previous results
  - In multi-user environments, allows users to leverage the results of each other's queries
- Data replication
  - End-to-end data replication rather than relying on RAID5 and other per-node availability mechanisms
- Streaming jobs where processing continues on new data as soon as it arrives
  - Partial-file support coming soon
- Dynamic creation of clusters
  - A simple config file describes the nodes participating in a computation. Different users or different tasks can use different sets of nodes.
- No privileged user access or software installation required.
  - A single Python script and non-root ssh access to each node is all that is required.
  - Low-bandwidth shared directory (e.g. NFS home dir) used for synchronization (not data storage) across nodes
- Don't re-invent the wheel
  - Thin layer on top of a POSIX (Linux, Unix, MacOS, etc.) environment.
  - Use OpenSSH for network communication and authentication
  - Use existing file systems & access control systems

**Learn More**: Examples | Installation

Last edited by mfisk, September 12, 2010