MP3 report

Algorithm

Our SDFS design consists of a master-slave model where one node is elected as the master.
The main idea of our SDFS is that each node contains a globally consistent index file, which stores all bookkeeping information related to file versions, nodes where file is stored, etc.
To put/get/delete file, the master is informed first. The master edits the global index file accordingly and and the new index file is propagated to all slave nodes.
We handle put/get/delete in a 2 step manner. For example in put, we first call an RPC that returns the nodes where the file should be sent. And then we make an RPC connection to each of the returned nodes and instruct the node to store that file. Similar logic is used for get/delete.
The load on our nodes stays balanced as we always choose a file to be placed on the node with least files.
Replication: We chose a replication value of 4. Therefore our SDFS will be able to handle upto 3 concurrent node failures. We write on 4 nodes always. For reads, we read from any one of the nodes. This ensures that our read always returns the latest write.
We used our MP2's failure detection system. All node failures and were detected with MP2's failure detector. MP1 was helpful when we wanted to grep logs of join, leave, failure from different VMs. Since it's hard to use a debugger when building distributed programs, we used MP1 to analyzing all the logs and pinpoint where the bug might be.

Analysis

(i) re-replication time and bandwidth upon a failure for a 40MB file

	Time/us	**Bandwidth/Mb**
1	507.658	160
2	491.944	160
3	56.372	160
4	439.087	160
5	440.145	160
Avg	387.041	160
SD	83.793	0

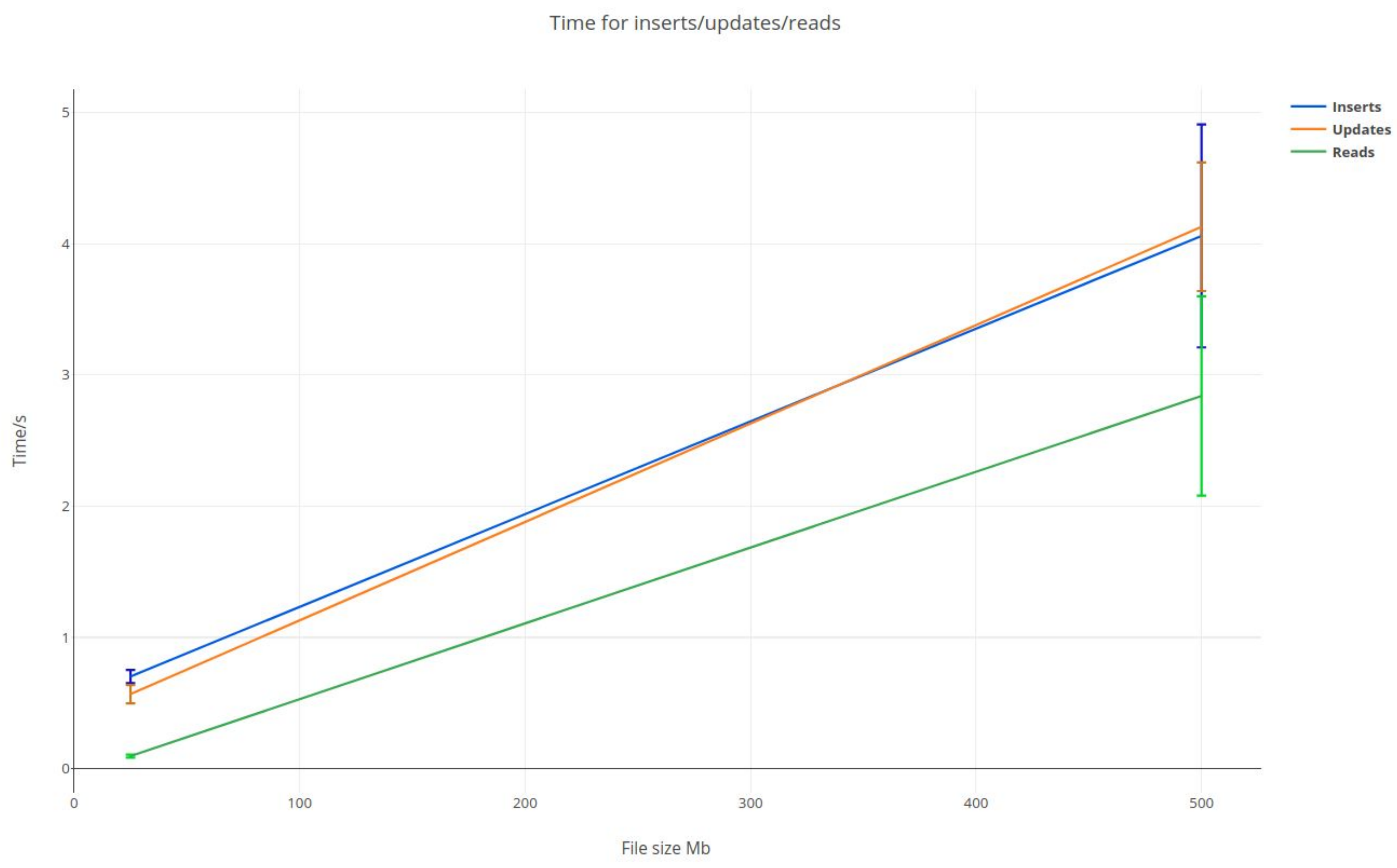
(iv) Time in second to store the Wikipedia corpus

	1	2	3	4	5	**Avg**	**Std Dev**
8 VMs	78.862	83.182	74.613	75.192	77.327	77.83	3.44
4 VMs	84.563	80.874	82.608	86.901	87.409	84.47	2.78

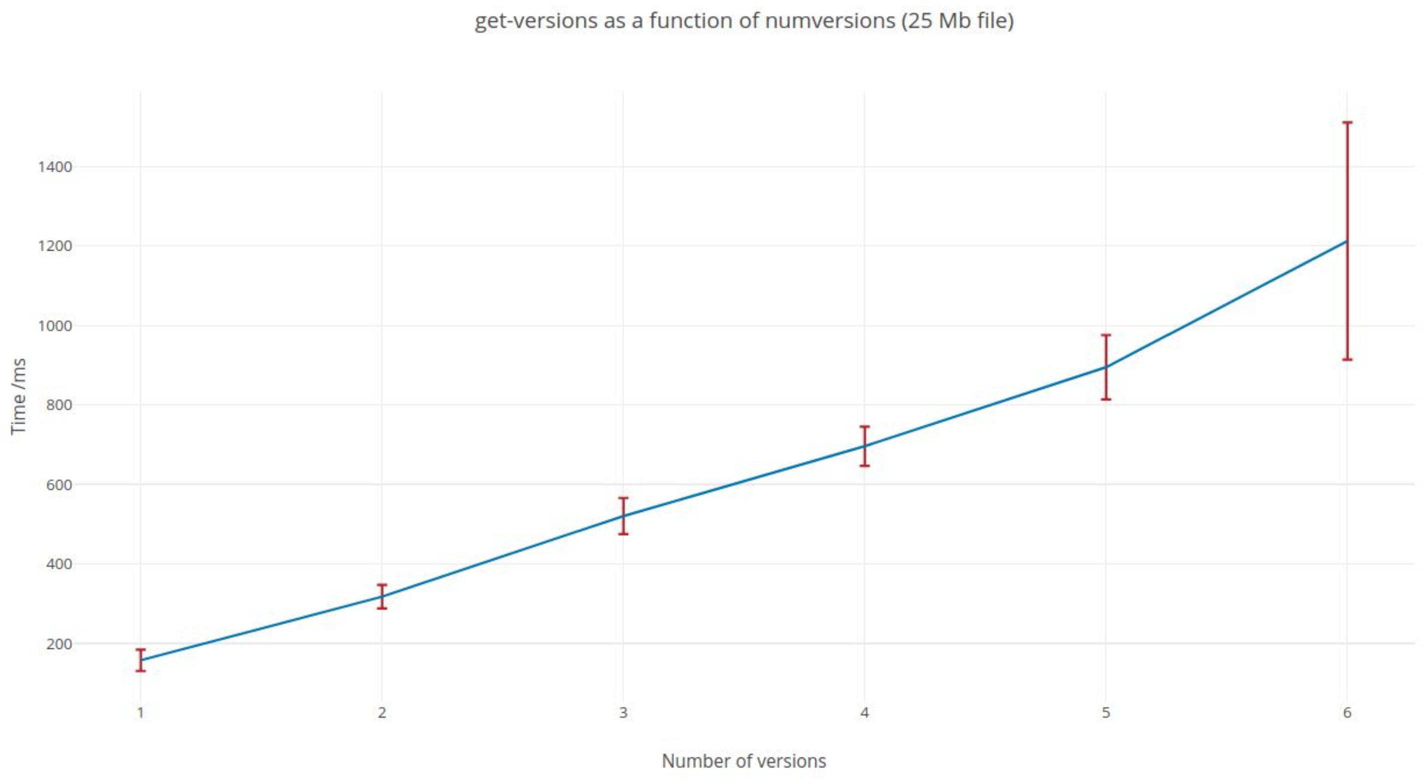
Our experiments showed a that the time to store the wikipedia corpus on 4 VMs was similar but slightly higher than for 8 VMS. The explanation for this is that as our design keeps 4 replicas and we choose the replica with the least number of files first, in the case of 4 VMs we have no choice but to store the data in the 4 VMs. With 8 VMs we have a choice and we could end up choosing nodes with less network latency.
The time for inserts/updates/reads increased with increase in filesize as expected and the time for inserts and updates was almost same and less for reads. Since *-put* is used to both insert and update the file and our logic is the same for both inserts and updates, the time taken was therefore similar aswell. Reads was faster as we read from 1 node where the file is stored where as we write to 4 nodes when we insert/update.

The time for get-versions increased with an increase in numVersions as greater the value of numVersions, greater the number of files and data needed to be fetched from different nodes, therefore incurring extra time.

(ii) Time for inserts/updates/reads



(iii) get-versions as a function of numversions (25 Mb file)


get-versions as a function of numversions (25 Mb file)
Time /ms
Number of versions