Spark

Background

Spark is a dataflow engine that is massively popular for performing batch computations. Its architecture has allowed it to demonstrate major performance improvements over MapReduce, a more naive solution to batch processing. Spark is extremely commonly used in industry, and as a result it is important to understand how it is able to achieve these performance boosts.

Problems with MapReduce

Recall: MapReduce is extremely inefficient for jobs that require many MapReduce calls/iterations (for example, page rank):

- Tons of disk I/O due to materializing intermediate state to HDFS
    - Writing the result of a MapReduce job
    - Reading the result of a previous MapReduce job
- Many redundant maps when not actually necessary

Resilient Distributed Datasets

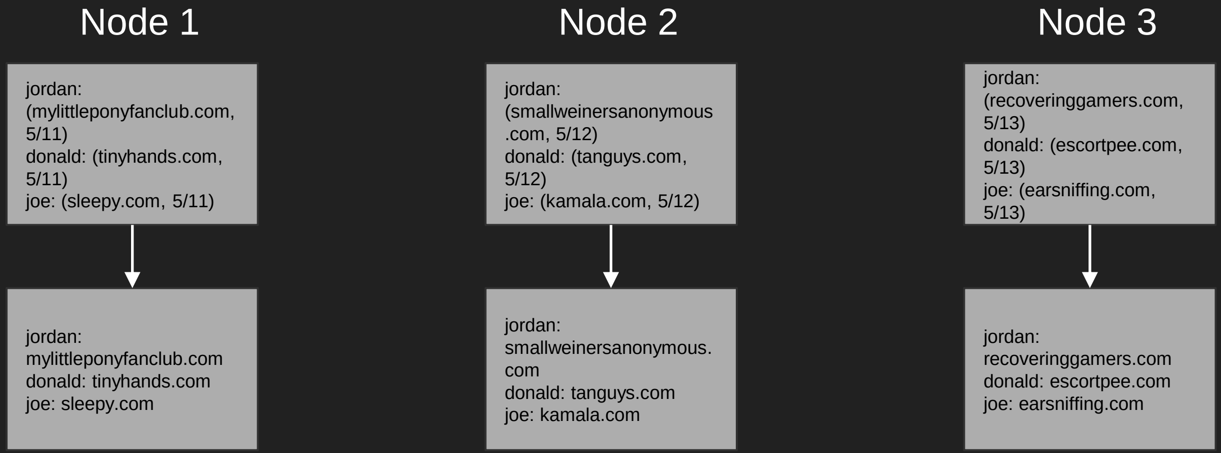
Instead of materializing intermediate state, Spark uses RDDs:

- In memory data structure representing the contents of a variable in a Spark program
    - In reality the data is probably held across multiple nodes in the cluster!
- Can create RDDs from data on disk or by using an operation on another RDD
- Spark keeps track of the lineage of each RDD
    - This means that it knows what computations needed to be performed to get the value contained in an RDD

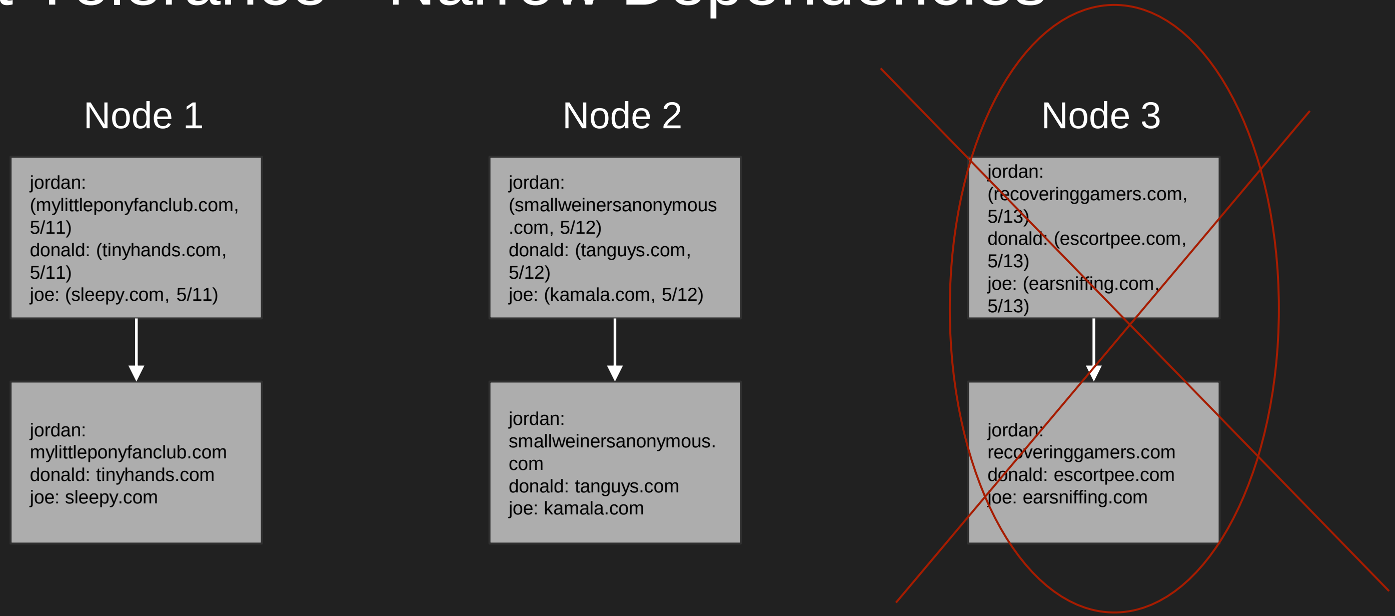
Fault Tolerance

Recall: If a MapReduce task fails, we can just resume from the output of the previous MapReduce calls. But Spark stores intermediate state in memory, and as a result we can’t just resume so easily. Instead, we actually have to do some recomputation, and recompute certain values based on their dependencies.

Fault Tolerance - Narrow Dependencies

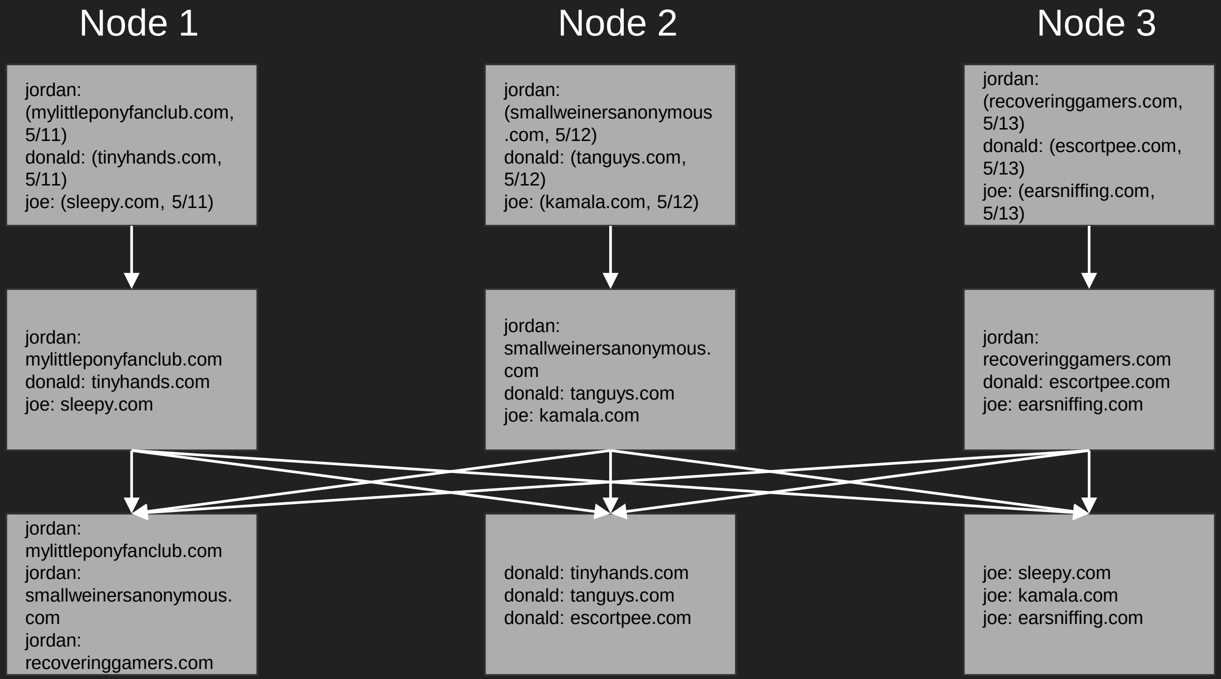

Node 1
jordan: (mylittleponyfanclub.com, 5/11)
donald: (tinyhands.com, 5/11)
joe: (sleepy.com, 5/11)
jordan: mylittleponyfanclub.com
donald: tinyhands.com
joe: sleepy.com
Node 2
jordan: (smallweinersanonymous.com, 5/12)
donald: (tanguys.com, 5/12)
joe: (kamala.com, 5/12)
jordan: smallweinersanonymous.com
donald: tanguys.com
joe: kamala.com
Node 3
jordan: (recoveringgamers.com, 5/13)
donald: (escortpee.com, 5/13)
joe: (earsniffing.com, 5/13)
jordan: recoveringgamers.com
donald: escortpee.com
joe: earsniffing.com

Fault Tolerance - Narrow Dependencies


Node 1
jordan: (mylittleponyfanclub.com, 5/11)
donald: (tinyhands.com, 5/11)
joe: (sleepy.com, 5/11)
jordan: mylittleponyfanclub.com
donald: tinyhands.com
joe: sleepy.com
Node 2
jordan: (smallweinersanonymous .com, 5/12)
donald: (tanguys.com, 5/12)
joe: (kamala.com, 5/12)
jordan: smallweinersanonymous. com
donald: tanguys.com
joe: kamala.com
Node 3
jordan: (recoveringgamers.com, 5/13)
donald: (escortpee.com, 5/13)
joe: (earsniffing.com, 5/13)
jordan: recoveringgamers.com
donald: escortpee.com
joe: earsniffing.com

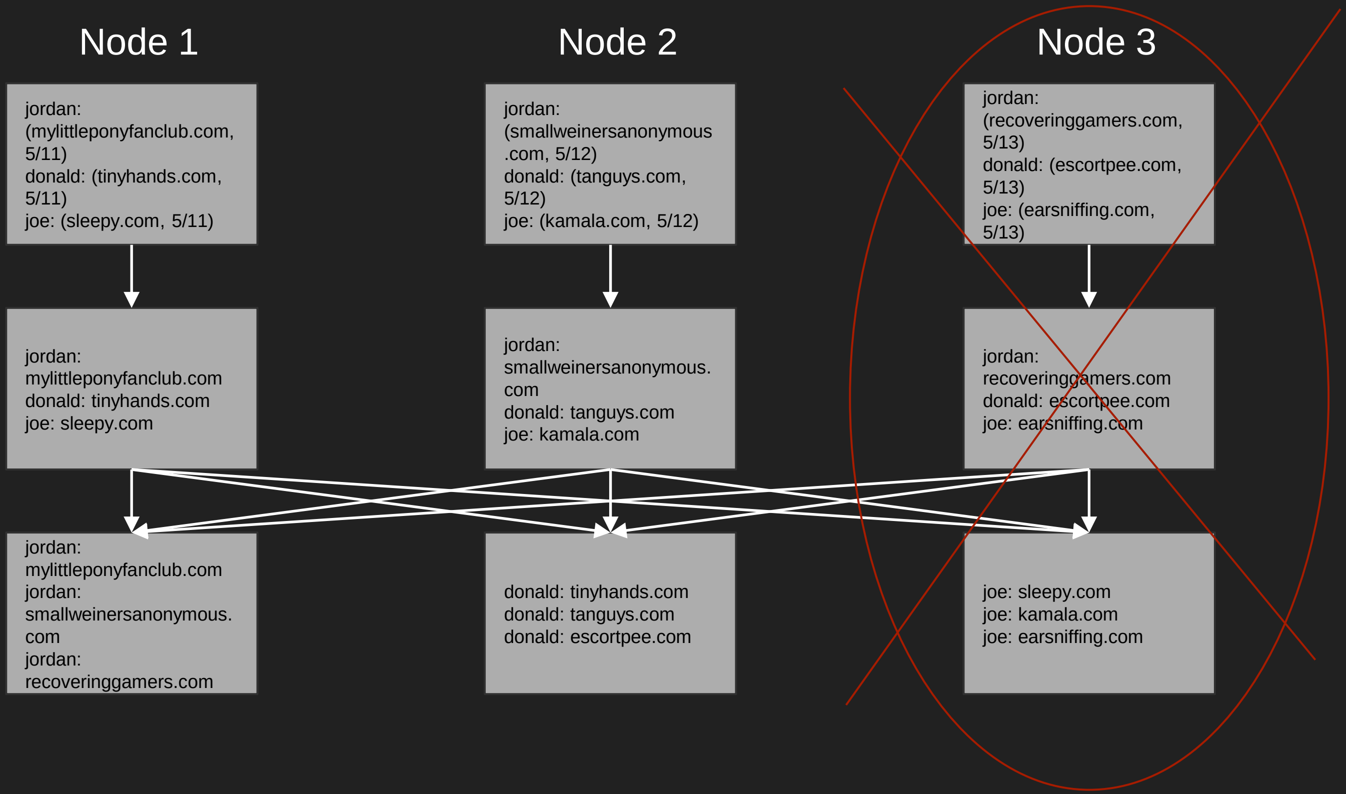
Narrow dependency: computation of each RDD local to one node. If node 3 fails and only has narrow dependencies, we can just re-perform its computations on node 1 and node 2, without having to actually alter the existing state on node 1 and node 2.

Fault Tolerance - Wide Dependencies



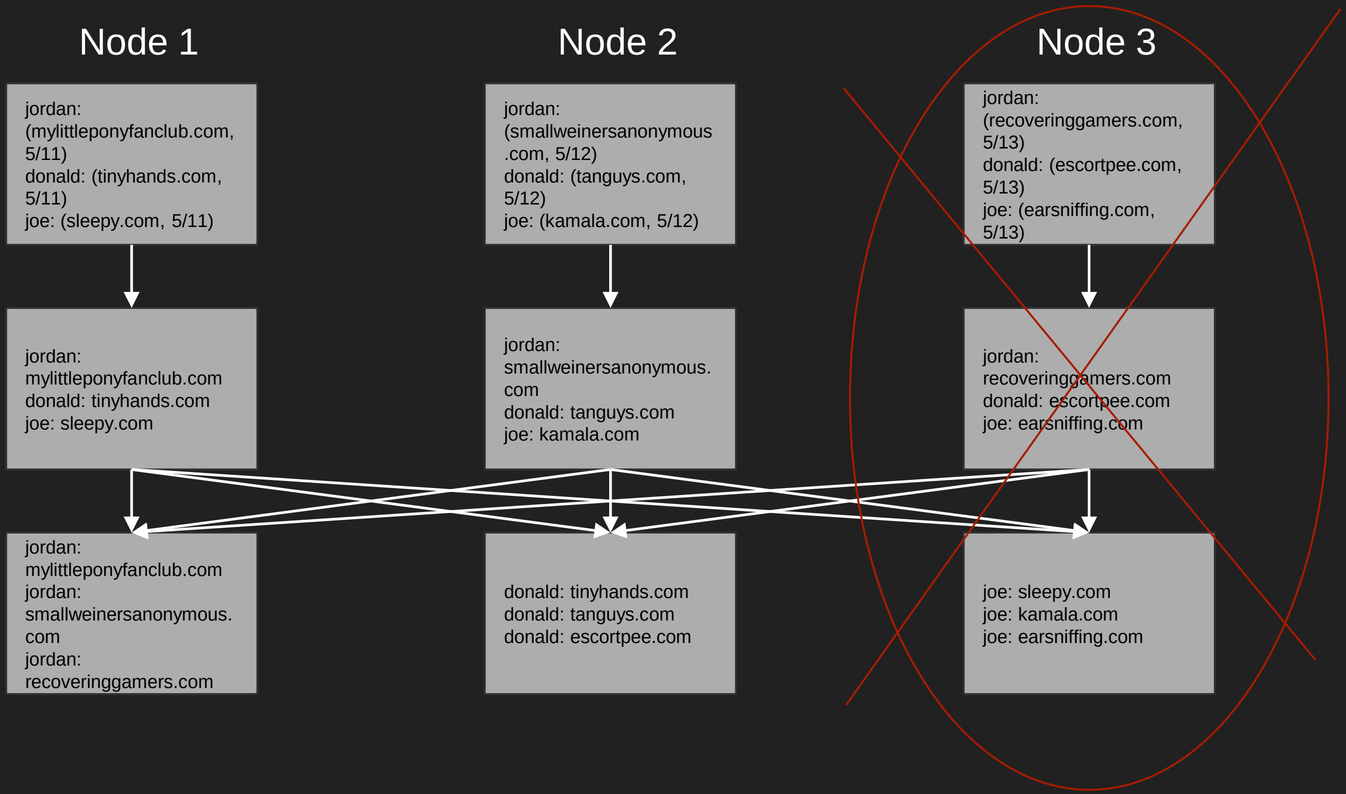
Node 1
Node 2
Node 3
jordan: (mylittleponyfanclub.com, 5/11)
donald: (tinyhands.com, 5/11)
joe: (sleepy.com, 5/11)
jordan: (smallweinersanonymous.com, 5/12)
donald: (tanguys.com, 5/12)
joe: (kamala.com, 5/12)
jordan: (recoveringgamers.com, 5/13)
donald: (escortpee.com, 5/13)
joe: (earsniffing.com, 5/13)
jordan: mylittleponyfanclub.com
donald: tinyhands.com
joe: sleepy.com
jordan: smallweinersanonymous.com
donald: tanguys.com
joe: kamala.com
jordan: recoveringgamers.com
donald: escortpee.com
joe: earsniffing.com
jordan: mylittleponyfanclub.com
jordan: smallweinersanonymous.com
jordan: recoveringgamers.com
donald: tinyhands.com
donald: tanguys.com
donald: escortpee.com
joe: sleepy.com
joe: kamala.com
joe: earsniffing.com

Fault Tolerance - Wide Dependencies


Node 1
Node 2
Node 3
jordan: (mylittleponyfanclub.com, 5/11)
donald: (tinyhands.com, 5/11)
joe: (sleepy.com, 5/11)
jordan: (smallweinersanonymous .com, 5/12)
donald: (tanguys.com, 5/12)
joe: (kamala.com, 5/12)
jordan: (recoveringgamers.com, 5/13)
donald: (escortpee.com, 5/13)
joe: (earsniffing.com, 5/13)
jordan: mylittleponyfanclub.com
donald: tinyhands.com
joe: sleepy.com
jordan: smallweinersanonymous. com
donald: tanguys.com
joe: kamala.com
jordan: recoveringgamers.com
donald: escortpee.com
joe: earsniffing.com
jordan: mylittleponyfanclub.com
jordan: smallweinersanonymous. com
jordan: recoveringgamers.com
donald: tinyhands.com
donald: tanguys.com
donald: escortpee.com
joe: sleepy.com
joe: kamala.com
joe: earsniffing.com

In this case, the final state of node 3 has wide dependencies in the sense that it relies on computations from node 1 and node 2 to be calculated. If node 3 crashes, we must rerun computation on both nodes 1 and 2 in order to regenerate this state. This takes much longer to do!!

Fault Tolerance - Wide Dependencies


Node 1
Node 2
Node 3
jordan: (mylittleponyfanclub.com, 5/11)
donald: (tinyhands.com, 5/11)
joe: (sleepy.com, 5/11)
jordan: (smallweinersanonymous .com, 5/12)
donald: (tanguys.com, 5/12)
joe: (kamala.com, 5/12)
jordan: (recoveringgamers.com, 5/13)
donald: (escortpee.com, 5/13)
joe: (earsniffing.com, 5/13)
jordan: mylittleponyfanclub.com
donald: tinyhands.com
joe: sleepy.com
jordan: smallweinersanonymous. com
donald: tanguys.com
joe: kamala.com
jordan: recoveringgamers.com
donald: escortpee.com
joe: earsniffing.com
jordan: mylittleponyfanclub.com
jordan: smallweinersanonymous. com
jordan: recoveringgamers.com
donald: tinyhands.com
donald: tanguys.com
donald: escortpee.com
joe: sleepy.com
joe: kamala.com
joe: earsniffing.com

In this case, the final state of node 3 has wide dependencies in the sense that it relies on computations from node 1 and node 2 to be calculated. If node 3 crashes, we must rerun computation on both nodes 1 and 2 in order to regenerate this state. This takes much longer to do!!

To avoid this situation, Spark allows developers to checkpoint certain application state to disk so that if node 3 were to crash, we would have the result of the wide dependency saved.

Conclusion

By better utilizing memory, and refraining from writing all state to disk, Spark is able to provide huge performance gains compared to MapReduce. While it comes at the cost of using more RAM, and having slightly worse fault tolerance than MapReduce, it seems that developers have realized that the increase in speed is well worth these tradeoffs, and MapReduce is seldom used in practice for large scale computation anymore.