15-440 Homework #3
Kesden/Fall 2013

Data-Intensive Scalable Computing (DISC)

1. How would it impact Hadoop's performance if it were to be implemented over AFS instead of HDFS? Why?

   Performance would be quite poor. For example, instead of dispatching the worker over the data via the location-aware file system, the location-transparent nature of AFS would force the data to be moved to the workers. Add inefficiencies due to cache size limits, block size limits, file size limits, etc, all intended for "normal" files not "Big Data" and it becomes a real mess.

2. The output of a Mapper is written into the local filesystem instead of the global filesystem. Why? Your answer should explain both why writing into the global file system would be undesirable as well as why it would be of minimal benefit.

   Writing to the local file system is more efficient as it doesn't involve network traffic. Writing into the distributed file system as we know it wouldn't be helpful, because the data isn't generally needed, but only needed to the specified reducers. One way to think of it is that the goal is to write it to the output of the mappers to the designated reducers and the local file system is little more than a buffer cache on route.

3. Why does Hadoop sort records en route to a Reducer? How would it affect things if these records were processed by the Reducer in the order in which they were received from the various Mappers?

   The Reducer needs the records sorted in order to bucket them. Unless the records are bucketed by key, they can't be reduced by key.

4. How is the failure of a Mapper or Reduce managed?

   The failed job is restarted elsewhere.

Distributed Computing Models

5. If processor allocation is optimal, is it possible that migration will subsequently improve system performance? If not, why not? If so, how?

   Possible (but unlikely). It is possible that the correct host for "right now" and the correct host for "later" are different, because some jobs end and others start, changing the available resources, changing the optimal allocation. If the gain of the move is bigger than the cost of the move – the migration is a win. Even with perfect foresight, migration could be a win in certain situations.

6. Why are periodic broadcast advertisements often considered to be a poor way of communicating information about resource availability? What is the risk?

   The information could become stale the moment it is broadcast. And, it isn't stall in one place – it is stale everywhere. The biggest risk is advertising availability globally. A bunch of hosts can dispatch work. The result of the "Thundering herd" of work is that the previously available resources are now overwhelmed. This problem can move from resource to resource – causing the herds to thunder, but little work to be accomplished.

7. Please explain two commonly used alternatives to the advertisements mentioned above and the relative costs and benefits.

   Those needing resources can poll for availability, which ensures that the information is current when used – even though it might require more traffic for multiple consumers to poll multiple hosts.

   A single coordinator can also be used to dispatch work, which reduces the need for communication and prevents thundering herds – but at the cost of a single pointof failure, a hotspot, etc.

8. When is a shared memory programming model typically a better approach than message passing? Why? Is anything special required or acutely helpful? Why?

   Shared memory is better when you have – shared memory. If not, message passing often provides a much cleaner paradigm. Of course, if the problem can't be structured such that message passing works, it might be necessary to build a shared memory with the necessary consistency guarantees: The weaker the guarantees, the better. And, the fewer structures that need to be shared, and the lower the contention for them, the better.

9. What type of tasks would you expect to benefit from Hadoop over OpenMPI? Why?

   Hadoop is better for "Big Data" problems, a.k.a. "Data-Intensive Scalable Computing (DISC), where the data can't be kept in memory, even across the distributed memory. This is because the Map-Reduce paradigm is designed to process the data with as little movement and communication as possible. By contrast, OpenMPI implementations are better in situations where the data is small and the processing upon it is big, such that the cost of communication is amortized over significant processing.

Distributed File Systems

10. In class we observed that AFS and NFS manage consistency differently. AFS issues callbacks upon updates. NFS validates the client cache periodically.

    (a) Do either of these mechanisms eliminate the window of vulnerability? If so, how? If not, is possible to eliminate the window of vulnerability? Why or why not?

    Nope. That window cannot be eliminated without locking to protect each use, which would be expensive, and (in the common file-system case) likely used very little, as most data is not likely to be changing and shared.

    (b) Which mechanism will result in less network traffic in the event that many dozens of clients have the same file open for high-frequency random-access reads?

    Given that modern implementations of NFS have client-side caches, there is likely to be little difference. Back in the day, NFS would have required traffic for each read and write.

11. Consider Coda's whole-file semantics, including the behavior of open() and close as well as of caching.

    (a) How does it enable disconnect and weakly connected operation, e.g. use of the file system even when network connectivity is poor or non-existent?

    By caching whole files, Coda is able to take advantage of spacial locality. Once the file si accessed – the rest of it will be available for use.

    (b) In what ways does it limit file system performance and capability?

The file system can't hold any file larger than the client-side cache, it can't have more files in use than the cache can hold, and the whole file needs to be read and written – even if only parts have been used.

Design: Special Purpose Distributed File Systems

12. Consider the design of a distributed file system for light-weight mobile devices, especially smart phones, such as common iPhone, BlackBerry, Android, and Windows Mobile devices.

    i. The file system should be robust without involving any off-line backups, e.g. tape.
    ii. It should view the device's storage as a cache for the actual data, but not necessarily the primary copy.
    iii. It should assume a workload similar to what we see with these devices now, e.g. notes, calendars, photos
    iv. It should support user data, not necessarily user programs
    v. It should facilitate the migration from one device to another and the use of the data on at least one host computer
    vi. User data should stay private to that user, but should be very quick to access, especially from the device, itself.

    Assume the following properties of the systems:

    i. Files are generally "small", e.g. text messages, notes, short documents, and cellphone photos, but not long hi-res videos, databases, etc.
    ii. Latency is very high, e.g. 500mS
    iii. Bandwidth is modest, but not terrible for downloads, e.g. 1Mbps
    iv. Bandwidth for uploads can be outright bad, e.g. 0.20 Mbps
    v. Although the data can be changed from multiple hosts, it will not be accessed concurrently, since there is only one user.
    vi. The storage on the each of the mobile device and host are large relative to the user's mobile needs
    vii. Files access generally requires the whole file, rather than random access to only some part of it.
    viii. Off-device storage is "Free"
    ix. The wired Internet is "Fast and wide"

(a) What are the most important challenges presented by these requirements?

The high latency is probably the biggest challenge. This is compounded by the low bandwidth, the most client capabilities and the need for migration. We are tempted to want a solution that keeps the data on the server, but that will be too latent and exhaust the bandwidth, or caches the data on the client, which doesn't have storage for it.

(b) Please describe the architecture of your solution, include especially descriptions of caching, replication, checkpointing, and the protection of privacy.

Answers vary.

Security

13. Consider *Onion Routing* and the case of a compromised router. In this worst case, will it know the source of the message, the destination of the message, both? Why?

    In the worst case, it will know the source or destination, but not both. This is because the route has to be longer than one hop, and therefore the worst case is that the compromised router is the first one or the last one.

14. Consider *Onion Routing,* why is the path chosen in advance by an agent of the client, rather than the network hop-by-hop?

    It can't safely be chosen hop-by-hop. Hop-by-hop would require that each hp know the destination. Otherwise, how does it get there? That ain't good!

15. Kerberos enables a client to communicate credentials to a server. What guarantees that the server will be able to trust these credentials?

    They are encrypted with a secret that the presenter doesn't have. As a result, the presenter can't decrypt them to change them and reencrypt them.

16. Kerberos uses *symmetric/secret key* cryptogrophy, rather than *asymmetric/public key* cryptopgraphy. Why?

    It is less computationally intensive and there is nothing to be gained by a public key system. The KDC couldn't encrypt with a public key, as the presenter could use it to decrypt, change, and re-encrypt the message. So, it would have to encrypt with the private key. This would make the message readable by others (with the public key), cost time, and gain nothing.