CMPSCI 691 Graduate Systems **Fall 2009**

Lecture 20

Lecturer: Emery Berger *Scribe: Dimitar Gochev*

20.1 Databases

There is a lot of discussion in the CS world on the following quesiton: Should we use databases, or should we use distributed hash tables with key-value pairs? This is commonly referred to as SQL vs NoSQL; however, frequently there is SQL layered on top of NoSQL.

SQL is a declarative language, high-level, clean and can be optimized. Language is equivalent to 1st order closure language. All queries are polynomial time by definition. Queries are taken care of by the query optimizer, which knows where the data can be found and can schedule jobs for getting the data. Stored procedures and database triggers - extensions to SQL that make it stateful and iterative. These can cause conflicts, and are difficult to debug. These are an example of "aspect-oriented programming" - moral equivalent of global variables, non-modular and goes against object-oriented programming model.

Terminology: Scale up (to a specific number of machines) vs. Scale out (to a large number of machines, example: Facebook) Database systems scale up to a cluster. Theoretically they can scale up more, but no such systems exist. Because MySQL does not scale, Facebook uses MySQL + mem-caching (one big lock, poorly engineered)

Insight behind column-based db's: queries usually don't need every part of a record, but rather only a certain field. There is no need to read all the fields from the disk when only one is wanted. Aside: Stonebreaker founded Vertica (column-based database company) so he has a financial interest in the comparison he presents in his paper. Databases follow the ACID reliability model: atomicity, consistency, isolation, durability.

20.2 MapReduce

MapReduce is NOT a database; its intended purpose is different. Types of workloads: DSS/DW (Decsion Support/Data Warehouse) - longer queries, higher read activity, data is static vs. OLTP (online transaction processing) - numerous short queries, higher write activity on volatile data – MapReduce is better at this type of workload

MapReduce is actually map-flatten-sort-reduce; the sort is the difficult step MR is fundamentally about fault-tolerance (extremely important issue with databases because machines will definitely fail) - handles all of the underlying issues such as congestion, lost packets, etc. (MapReduce is significantly faster than Hadoop because MapReduce is written in C++, and Hadoop is in Java, which has inherent inefficiencies - string operations are expensive for example, because strings are immutable in Java)

20.3 What is the next step forward? Why are database people not focusing on fault-tolerance?

Databases didn't scale up, which lead to work in hiding parallelism, caching, MapReduce, etc. Eventually the model might be redesigned, and the consistency guarantees reduced. (Eventual consistency) It's shocking that there is still money in infrastructure, because it is available for free. Hardware is the bigger concern in data centers and issues like energy efficiency are coming up more and more.