Thoughts about various papers I have read

1. Dremel: Interactive Analysis of Web-Scale Datasets

- uses a serving tree in order to answer queries
- unlike Hive/Pig it executes queries natively without translating them into MR jobs
- uses a columnar storage format for nested data

- it works well because usually the data does not contain values for all the fields in a protocol buffer structure

2. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters

- uses k-means in order to classify jobs into a set of categories
- each job is characterized by time in seconds, CPU usage in cores and memory usage in gigabytes
- this may be a good source to evaluate using jobs similar with those in industry

3. FlumeJava: Easy, Efficient Data-Parallel Pipelines

- couple of classes to express parallel computation
- evaluation is deferred and FlumeJava constructs an execution dataflow graph
- when results are needed the graph is optimized and tasks are completed using MR jobs
- it automatically decides if an operation should be implemented as a sequential loop or as a MR job according to the size of the data
- PCollection<T>, PTable<K, V>, PObject<T>
- parallelDo(), groupByKey(), combineValues(), flatten()
- PObjects can be either deferred or materialized

4. Piccolo: Building Fast, Distributed Programs with Partitioned Tables

- key-value storage partitioned over multiple machines
- computation organized as a series of application kernel functions that are launched on multiple nodes
- control functions create shared tables, launch multiple kernel function and perform global synchronization
- user-defined function to solve concurrent updates on a key
- uses Chandy-Lamport snapshot algorithm together with a log in order to provide fault tolerance
- data store in-memory which basically means that it is difficult to scale

5. Tenzing: A SQL Implementation On The MapReduce Framework

- they have optimized MapReduce so that they get better performance
- relaxed the requirement of all values for the same key to end up in the same reduce call. Instead, they can just end up in the same reduce shard => no sorting step => implemented a hash table based aggregation
- MR streaming: the reducer of the upstream MR and the mapper of the downstream MR are co-located in the same process

6. Nova: Continuous Pig/Hadoop Workflows

- a graph of interconnected Pig Latin programs
- a model of task and channels. Channels are data containers and tasks are processing steps
- it supports vertexes between channels and tasks
- types of vertexes: all (reads a complete snapshot of the data from a given input), new

(only reads data that is new since last invocation), B (emits a new full snapshot), delta (emints only the diff)
- supports data-based triggers, time-based triggers and cascade triggers
- at each step it writes into HDFS, thus it is not really streaming!
- scan-sharing: if two tasks depend on the same channel then they share the scan

7. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

- bla bla

8. Spark: Cluster Computing with Working Sets

- resilient distributed datasets: a read-only collection of objects partitioned across a set of machines that can be rebuild if a partition is lost
- supports several operations: reduce, collect, foreach

9. Scripting the Cloud With Skywriting

- a dynamically-typed, purely-functional language
- scripts can create tasks asynchronously that can further spawn other tasks => jobs can dynamically grow
- it uses the idea of references that are only instantiated when needed or via the dereferencing operator (*)

10. CIEL: A Universal Execution Engine for Distributed Data-flow Computing

- can memoise the results of tasks and supports the streaming of data between concurrently-executing tasks
- data is stored on workers. Fault-tolerance is provided by re-doing the computation
- stream consumers are implemented using a thread that continuously reads

11. A Simple totally ordered broadcast protocol

- describes Zab the protocol behind ZooKeeper

12. ZooKeeper: Wait-free Coordination for Internet-scale Systems

- all the write requests go to the master and the read requests go to the replicas
- data should be lightweight and it is stored in a file hierarchy type of znodes
- znodes are of two types: regular and ephemeral
- consistent hashing may not be needed to observe a set of coordinators

13. HaLoop: Efficient Iterative Data Processing on Large Clusters

- stores invariants on local machine
- stores reduce output on local machine
- does some caching and tries to assing the next map/reduce tasks on the nodes where the data is stored

14. MapReduce Online

- map and reduce tasks are assigned when a map step starts it opens a TCP socket to the reduce task
- subsequently it improves the mechanism by writing into files that later on are read by the reduce task
- adds supports for streams by sending data on the fly and by starting downstream computations as certain progress levels are reached
- however, it only works for reduce tasks that are distributive or algebraic aggregates, otherwise the computation has to be restarted every single time
- **I should use it in the evaluation of my system**

15. Kepler + Hadoop: A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems

- Kepler adopts the actor-oriented modeling. Each actor can do a atomic or composite operation. Composite actors (sub-workflows) are composed of atomic actors
- integrates Hadoop into Kepler a worflow manager
- the language seems to be very basic

16. Session Guarantees for Weakly Consistent Replicated Data

- intents to present individual applications with a view of the database that is consistent with their own actions, even if they read and write from various servers
- proposes four guarantees:
    - read your writes - read operations reflect previous writes (use sessions to guarantee it)
    - monotonic reads - successive reads reflect a non-decreasing set of writes
    - writes follow reads - writes are propagated after reads on which they depend
    - monotonic writes - writes are propagated after writes that logically precede them
- a session maintains a read-set and a write-set of WIDs for the writes that are relevant to sesions reads, respective writes
- introduces version vectors in order to implement the four guarantees
- wids are defined as (server,clock) where clock is specific for each server
- let V[S] = the time of the latest WID assigned by server S
- to check if ones set of WIDs is a subset of another check if V1 dominates V2 (i.e. if every element if greater or equal to its correspondent in the other array)

17. Transactional Storage for Geo-replicated Systems

- key-value store that supports transactions and replicates data across distant sites
- replicates data asynchronously while providing strong guarantees within each site
- precludes write-write conflicts (as the ones present in Picollo)
- conflicts are avoided using preferred sites and csets(counting sets)
- relaxes ordering constraints: if A uses site 1 and B uses site 2 then A may see a different transaction order than the one seen by B
- preferred site of an object: where writes to the object can be committed without checking other sites for conflicts
- I will have to finish reading this paper

18. Pregel: A System for Large-Scale Graph Processing

- computations consist of a series of iterations called supersteps. During a superstep the framework invokes a user-defined function for each vertex. The function specified the behaviour at vertex V at a superstep S. The vertex can read messages sent at superstep S - 1 and send messages to other vertices that will be available at superstep S + 1. On top of that it can modify the state of V.
- the computation is conducted in a master/worker pool way. Each worker receives a partition of the graph based on Vertex IDs. Before each superstep the worker takes a snapshot of its state and proceeds to do the computation. When it finishes it tells the master how many vertices will be active in the next superstep
- It points to a paper: A higher order estimate of the optimum checkpoint interval for restart dumps. Good to read if I plan to use checkpointing
- most master operations (input, output, computation) are terminated at barries. It may be interesting to think if we can eliminate the barrier

19. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

- a user specifies an arbitrary directed acyclic graph to describe the application's communication patters and express the data transport mechanisms (files, TCP pipes and shared-memory FIFOs)
- graph vertices can have arbitrary number of inputs and outputs
- an implementation for each type of vertex must be provided
- a series of operators are provided to connect vertices (replicate graph, directly connect, bipartitate graph, merge two graphs)
- failure policy assumes that all the vertex computations are deterministic
- various optimizations can be made on the graph (.e.g add an intermediate aggregation layer that can improve data-locality and reduce the amount of data sent between racks)

20. The Design of the Borealis Stream Processing Engine

- their goals are to support: dynamic revision of query result and dynamic query modification
- streams are modeled as append-only tuples of form (k1,...,km,a1,...,an) where k1...km comprise a key for the stream and a1,...,am provide attribute values
- supports three other kinds of stream messages (i.e. tuples): insertion messages, deletion messages, replacement messages
- I'll have to read again this paper, however it does not seem to be a good paper

21. Flux: An Adaptive Partitioning Operator for Continuous Query Systems

- a dataflow operator that encapsulates adaptive state partitioning and dataflow routing. Being adaptive is a requirement because the throughput of streams care vary a lot. Thus, the system must be able to balance itself as the data increases
- one can try to partition operators horizontally. However, a slowdown in one of the machine in which a partitioned operator is running can affect all the other partitioned operators (i.e. a dataflow pipeline is as fast as its slowest operator)

22. Interpreting the Data: Parallel Analysis with Sawzall

- it is mainly suited for structured data on which we want to run queries that do not require extensive calculation
- computation broken in two phases: evaluate the analysis on each record individually and aggregate the results
- the first phase is expressed in a new procedural language that executes one record at a time in isolation. The second phase is restricted to a set of predefined aggregators. In this the system can obtain better throughput
- they assume that the order in which the records are processed is unimportant
- they also constrain the aggregation operations to be commutative and associative so that intermediate values can be in an arbitrary order and grouped together
- it operates on each input record individually and it can emit zero or more intermediate values. These records are sent to the nodes running the aggregators, which collate and reduce the intermediate values. The final step collects and collates data from each aggregator into a single file
- an aggregator can be indexed, which creates a distinct individual aggregator for each unique value of the index
- supports job chaining (pipelining). However, a job must completely finish before its dependants can start

23. A Tool for Prioritizing DAGMan Jobs and Its Evaluation

- the major impediment to scheduling complex computations efficiently is temporal unpredictability because jobs are executed at different sites
- schedule jobs so that the number of jobs that are eligible for assignment to remote workers is maximized
- scheduling conducted in three phases: divide, recurse, compose
- divide
    - **we remove from the DAG every shortcut arc (i.e. u->v, such that v can be reached from u without using the arc)**
    - **we decompose the DAG into maximal connected bipartite dags**
- recurse
    - **we attempt to find an IC-optimal schedule (increases jobs eligible) for each component**
- combine
    - **we investigate priorities among building blocks**
    - **check if DAG respects inter-building block priorities**
    - **topological sort components according to the priority relation**
- Now the heuristic algorithm that works for all DAGs:
    - step 1 is the same
    - let C(S) be the smallest subgraph of G with the following properties: it contains a job s, if C(S) contains a source of G then it contains every child of that source, if C(S) contains a job then it contains every parent of the job
    - decompose G into C(S)s
    - check if each C(S) is isomorphic to a bipartite DAG with a know IC-optimal scheduler. If not, then we produce a schedule using a heuristic that executes jobs in the order of the job-outdegree
    - …
- I wonder if maximizing the number of available jobs is the best scheduling?
- has a good set of DAGs to test a system
- a DAG of 3000 nodes takes ~16 seconds to optimize and one with 48000 took 845 seconds

- when batches of requests arive rather often, or when batch sizes are either small or large, theer is little difference between FIFO and PRIO-scheduling

24. A Survey of Data Provenance in e-Science

- the two major approaches to representing provenance information use either annotations or inversion. Annotations are metadata comprising of the derivation history of a data product. On the other hand, inversion assumes that some derivations can be inverted so that one can find out the input was supplied to them
- users can search for all datasets generated by a specific workflow
- most of the survey presents general workflow lineage capturing systems (May use it to show that Dron is something new)

25. Wing for Pegasus: Creating Large-Scale Scientific Applications

- uses semantic representations to describe compactly complex computations in a data-independent manner
- 3 stages in the creation of workflows
    - create workflow templates, specify high-level structure of the workflow in a data-independent representation
    - create workflow instances which specify what data is to be used in the computation
    - create executable worflow which specifies the data replicas to be used and where the computation will occur
- workflow templates and instances are semantic objects whose components, data requirements and data products are represented
- Wings provides the features described above
- maybe go again over their language description

26. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World

- continuously adaptive query processing
- in stream query processor the arrival of data can be extremely high or bursty, detailed statistics of data are not readily available, time or order may be important
- CQ - usually have a monitoring or a filtering aspect
- processing each query individually can be slow and wasteful of resources, as the queries are likely to have some commonality => shared processing must be a fundamental capability of the system
- consists of a set of composable dataflow modules or operators that produce and consume records
- Modules:
    - Ingress and Caching: responsible for interfering with external data sources
    - Query Processing: pipelined, non-blocking versions of standard relational operators such as joins, selections, projections
    - Adaptive Routing: Eddies - adaptively decide how to route data on tuple-by-tuple basis. Juggle - perform online reordering for prioritizing records by content. Flux routes tuples to support parallelism and fault-tolerance
- Fjords - an inter-module communication API. It binds the various modules together. It provides a blocking enqueue or a non-blocking one. Thus, modules can decide if they want to block or not

27. Large-scale Incremental Processing Using Distributed Transactions and Notifications

- Percolator - a system for incrementally processing updates to a large data set
- 2 abstractions for performing incremental processing at large scale: ACID transactions over a random-access repository and observers as a way to organize incremental computation
- relaxed latency requirements allowed the designers to adopt a lazy approach in cleaning locks of failed transactions. Moreover, it allowed them not to implement a global deadlock detector
- written on top of Bigtable. It provides new features such as multirow transactions and the observer framework
- it depends on a timestamp oracle that provides strictly increasing timestamps
- also depends on a lightweight lock service to make the search for dirty notifications more efficient
- uses BigTable to provide random access => the web index can be updated instead of being recomputed

28. Quincy: Fair Scheduling for Distributed Computing Clusters

- graph-based framework for cluster scheduling under a fine-grain cluster resource-sharing model with locality constraints
- demonstrate mapping between the fair-scheduling problem and min-cost flow problem in a directed graph
- each job has its own DAG. Thus for each job the single centralized scheduling service is assigning a root task
- each root task submits worker tasks to the scheduler according to a job-specific workflow. The scheduler determines which tasks should be active and matches them with available computers
- the root task computes for each worker computer the amount of data a job would have to read across the network. Then it constructs two lists of workers and ranks that store more than a fraction of data
- starts from the idea that every scheduling decision can be quantified (e.g. data transfer cost incurred for running a task on a particular node, cost in wasted time in killing a task)
- Scheduling as a min-flow problem!!!

29. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language

- automatically translates data-parallel portions of the program into a distributed execution plan which is passed to the Dryad execution platform
- LINQ - Language INtegrated Query - a set of .NET constructs for programming with datasets
- LINQ
    - IEnumerable<T> - abstract dataset of objects of type T
    - IQueryable<T> - an (unevaluated) expression constructed by combining LINQ datasets using LINQ operators
- DryadLINQ composes all LINQ expressions into IQueryable objects and defers evaluation until the result is needed, at which point the graph is optimized
- metadata is used with the DryadTable so that the framework can optimize computation

30. Query Processing, Resource Management, and Approximation in a Data Stream Management System

- Data Stream Management Systems are expected to handle multiple continuous, unbounded, time-varying data streams
- a stream is mapped into a relation by applying a window specification
- relations are mapped to streams by using the operators:
    - Istream (input stream) applied to relation R contains a stream (t, s) whenever tuple s is in R at time t and not in R at time t - 1
    - Dstream (delete stream) - (t, s) is in the stream whenever (t, s) was in S at time t -1 and is not at time t

31. Nectar: Automatic Management of Data and Computation in Datacenters

- derived datasets, which are the results of computations, are uniquely identified by the programs that produce them
- automatically managed by a datacenter wide caching service
- 50% of the files were not accessed in the last 250 days
- data that has not been accessed for a long period may be removed and replaced with the computation that produced it
- instead of executing a user's program, Nectar can partially or fully substitute the results of that computation with data already present in the datacenter
- **efficient space utilization, reuse of shared sub-computations, incremental computations**(reuse results for old data)**, ease of content management**(little need for developers to manage their data)
- takes a DryadLINQ program as input and consults the cache system in order to rewrite the job
- treats all prefix sub-expressions of the LINQ expression tree as candidates for caching

32. BOOM Analytics: Exploring Data-Centric, Declarative Programming for the Cloud

- hypothesis: distributed systems benefit substantially from a data-centric design style (sets, relations, streams...). The key behaviour of such systems can be easily implemented using declarative programming languages that manipulate these collections
- Datalog
    - defined over relational tables
    - purely logical query language that makes no changes to the stored tables
- atomic Datalog timestamp
    - inbound events are converted into tuple insertions and deletions on the local table partitions
    - interprets the local rules and tuples according to Datalog semantics
    - updated to local state are made durable and outbound events are emitted