A survey on NoSQL and NewSQL datastores

Brecht Gosselé

***Abstract*—Advances in bioinformatics have driven down the cost of genome sequencing dramatically over the past years. This raises the question how to efficiently and flexibly handle and store the large amount of data this process generates. Similar developments in web technology have spawned numerous NoSQL and NewSQL datastores in the last decade, which offer robust, highly scalable distributed storage and flexible data modelling. This paper reviews 6 such datastores and compares them on characteristics relevant to high performance computing in general and bioinformatics more specifically.**

I. Introduction

Scientific progress has caused the cost of genome sequencing to drop at an exponential rate over the past decade and a half, even outpacing Moore's law since 2008 [28]. Because in all sorts of biological, medical and pharmaceutical research, more and more genomes are being sequenced, the amount of generated data increases rapidly. For instance, the whole genome sequencing pipeline of the Broad Institute [27], a reference in the field, generates in the order of 3 TB of intermediary data when sequencing a single human genome. Also, when sequenced with 50x mean coverage (denoting the average number of times a base has been read during the process [26]) a single human genome takes 50 GB to store in compressed format[1], so as this scales to genomes of millions of people and other organisms, storage and processing requirements become ever more demanding in terms of scalability, latency and concurrency.

A logical evolution has been to tackle these problems with high performance computing systems. The Exascience Life Lab of imec, Intel, Janssen Pharmaceutica and 5 Flemish universities, actively researches the application of supercomputers for accelerating the processing of whole genome sequences [25] [24].

Increasing popularity of web services such as social networks has caused a similar explosion of data for web companies. To handle this so-called Big Data [32] in an adequate way, traditional relational DBMS no longer suffice. Therefore, large web companies such as Google and Amazon have developped new storage solutions that meet the demands for high and incremental scalability, low latency and high availability [1]. This has spawned many so called NoSQL ('Not only SQL') databases, which loosen the rigid relational datamodel in favour of better scaling and easier distribution of the data.

NoSQL datastores come in multiple flavours and can be divided in a few categories based on the datamodel they use:

- Key-value stores: much like dictionaries and associative maps, these map unique keys to values. Values are uninterpreted byte arrays, and the only way to access them is by their key. This means modeling relations and complex structures between data, rich querying and indexing is not possible [23] [22].
- Columnar stores: based on the datamodel pioneered by Google's BigTable, these store data in "a sparse, distributed, persistent multidimensional sorted map" [5]. In BigTable's case this is a map of row key, column key and a timestamp. In this way, multiple versions of the data can be stored in chronological order. Because the system doesn't interpret the data, relationships can't be modeled. This is left to application logic [23].
- Document stores: these store data as key-value pairs encapsulated in documents. Values can be of a wide variety of types, such as nested documents, lists or scalar values. Attribute names can be dynamically specified at runtime and need not adhere to a fixed schema [4]. This is well suited for modelling complex data structures. Many document stores use the JSON-fileformat (or some derived form). In contrast to columnar stores, the values in documents are not opaque to the system and can thus be queried and indexed [23].
- Graph databases: as the name suggests, these originate from graph theory and use graphs as their data model. They are especially useful to manage highly interconnected data coming from sources such as social networks or location based services, replacing costly operations like recursive joins with efficient graph traversals [23].

The term NewSQL data stores is being used to classify a set of solutions that aim to combine the scalability, distribution and fault tolerance of NoSQL stores with the relational data model. Though they all use the relational model and supply SQL-querying capacities, NewSQL stores vary greatly under the hood, depending on the architecture they are built on [22].

This paper proceeds first by elaborating on the methodology used to select and compare the chosen databases and then by reviewing each of the six datastores in detail.

II. Methodology

Because of the enormous choice of NoSQL and NewSQL datastores, an exhaustive study wasn't feasible. This survey reviews the most popular datastores in a few relevant categories, namely document stores, wide columnar stores and NewSQL stores. Key-value stores and graph databases were not taken into consideration, as their respective data models are not suited for the application at hand. A selection was then made based on similar criteria as in [22], following the ranking of DB-Engine Ranking [39] as an indicator of popularity.

[1]This information was gathered during conversations with researchers at the Exascience Life Lab

	MongoDB	CouchBase Server	Cassandra	HBase	Cloudera Impala	VoltDB
Type	Document-store (BSON)	Document-store (JSON)	Wide columnar store	Wide columnar store	SQL on top of Hadoop	In-memory relational NewSQL
Querying & API	Proprietary language, dynamic queries with JS, rich API, MapReduce	N1QL, memcached API, MapReduce	CQL, rich API, MapReduce	No query language (Hive via workaround), Java API, MapReduce	SQL-92, MapReduce	SQL-92, Java stored procedures, rich API, MapReduce
Distributed	Master-slave, asynchronous replication	Multi-master, asynchronous replication	Masterless, asynchronous replication	Master-slave or multi-master, asynchronous replication.	Masterless	Masterless, updates executed on all replicas at the same time.
Storage	Standard FS's, 16MB document limit	Standard Unix & Windows FS's	Cassandra File System (HDFS compatible)	HDFS	HDFS or HBase	Main memory
Indexing	Primary & secondary on every attribute, B-tree	Primary & secondary, B-tree	Primary & secondary, LSM-tree	Primary & secondary, LSM-tree	Primary & secondary	Primary & secondary. Hash- & tree-indexes
Query optimization	Query optimizer, shard-keys to speed up distributed queries	memcached	Bloom filter	Bloom filter	Partition pruning using partition key	Queries in stored procedures planned at compile time
Partitioning	Range partitioning based on shard key	Hashing function	Consistent hashing	Range-partitioning	By default not, but possible	Consistent hashing
Consistency	Configurable	Within cluster: strong; within multiple clusters: eventual	Configurable	Strong	Relies on underlying storage layer	Strong
Concurrency control	Atomic single document operations, otherwise 2-phase commit; Concurrent reads, exclusive writes (lock on DB level)	Application can implement optimistic (using CAS) or pessimistic concurrency control	Row-level atomicity, CAS	Single-row transactions (ACID possible), OCC with MVCC for wider scope operations	Relies on underlying storage layer	ACID + data access serialized and executed in single-threaded environment
Open-source?	Y	Y	Y	Y	Y	Y

Table 1: An overview of the compared datastores and their features

This ranking tries to measure popularity based on a few parameters, such as the number of mentions on Web sites, general interest according to Google Trends, frequency of technical discussions on the Web, number of job listings, and number of professional profiles in which the systems are mentioned. The resulting selection consists of the document stores MongoDB and CouchBase Server, wide columnar stores Cassandra and HBase and NewSQL database VoltDB. There already exists an extension of the DNA sequencing pipeline used in the ExaScience Life Lab that allows for using MongoDB databases as input and/or output targets, making MongoDB even more relevant [8]. Lastly, NewSQL query engine Cloudera Impala was also taken into consideration because of explicit interest from researchers in the aforementioned lab. These 6 systems were then compared on characteristics relevant to high performance computing, such as indexing mechanisms, client interfaces to the data, distribution strategy, concurrency control and consistency models.

III. Document stores

A. MongoDB

MongoDB stores data in BSON (binary JSON) documents. It has powerful indexing support, with the ability to define secondary indexes of a wide array of types on all attributes, much like in the traditional relational model. These are implemented using B-trees [34]. To aid with denormalization, the documents can contain embedded documents and arrays, thus obviating the need for joins in the query language.
The document size is limited to 16 MB, to help ensure that a single document cannot take up excessive amounts of RAM or bandwidth. To store and retrieve larger files, the built-in tool GridFS (which is not an actual file system) can split up files in smaller chunks and store these chunks as separate documents [33].
MongoDB offers API's in many languages and the functionality to define the equivalent of SQL WHERE-clauses as javascript expressions. These are then translated to MongoDB's proprietary internal querying language [22]. The MongoDB query optimizer processes queries and chooses the most efficient plan for a query given the available indexes. These plans are cached if there are multiple viable options and can be reevaluated as the data evolves [35].
The consistency of MongoDB is configurable. Strong consistency can be attained in two ways: setting the connection to read-only from the master node (which has the most up-to-date version of the data), or forcing a write to succeed only after all replicas have acknowledged it. The former degrades the scaling ability of read requests, the latter the latency of write requests [22].
Data is replicated asynchronously using range-partitioning: nodes are responsible for ranges of keys. This speeds up range queries, but can create hotspots and load-balancing issues. To route updates to the right replicas, MongoDB operates in a master-slave setting.
For concurrency control, MongoDB offers single-document atomicity and implements reader-writer locks. Having to lock on writes severely impacts performance in write-intensive scenarios.

As a wrap-up, MongoDB stores BSON-files accessible through many API's with flexible querying and indexing techniques, but its concurrency and consistency model have some drawbacks.

B. CouchBase Server

CouchBase, result of the merger of CouchDB and Membase, stores data in JSON documents. It uses the memcached protocol for distributed caching and is intended for highly interactive applications with low-latency requirements [22] [9].
The JSON documents can be nested and are queryable through a SQL-like language, N1QL (note: the most recent version at the time of writing, released in March 2014, is still a developer preview) [11]. Primary and secondary indexes can be defined on the data and are implemented using B-trees [10].
Within one cluster, transactions are strongly consistent, but between multiple clusters only eventually consistent.
Couchbase lets clients choose between optimistic (using compare-and-swap) and pessimistic (using 'finegrained locking') concurrency control.

With its flexible data modelling, caching and concurrency control, Couchbase is a good fit for applications requiring fast and intensive interaction between client and data.

IV. Columnar stores

A. Cassandra

Apache Cassandra, originally developped at Facebook but later open-sourced, combines the data-model of Google's BigTable system with the architecture and distribution strategy of Amazon's DynamoDB. It is intended for flexible, highly-available storage of very large datasets, running on cheap commodity hardware and offering high write throughput while not sacrificing read efficiency [30].
Since its inception Cassandra has however diverged slightly from the BigTable data model. It now provides tables and composite columns -much like in a conventional schema, and comes with its own query language, CQL [31]. CQL resembles SQL in many ways, albeit with some restrictions. For instance, it doesn't feature the JOIN clause [13]. It strongly encourages physically collocating data that will be queried together, and supports denormalization with features such as collection types.
Cassandra has primary and secondary indexing mechanisms and implements these - like BigTable [5], using log-structured merge trees (or LSM trees). These allow for deferring updates and flushing them to disk in batches as soon as enough have accumulated, thus reducing disk I/O. This greatly benefits write throughput [37] [38] [30]. Cassandra, like BigTable, also offers Bloom filters [36]. These are an efficient probabilistic mechanism to predict whether an item is in a set (in this case, whether a key is in a table) and can thus significantly reduce unnecessary table scans [30].

In order to scale linearly in the number of nodes to very large datasets, Cassandra operates in a fully masterless fashion. In terms of the CAP-theorem, it focusses on availability and partition-tolerance, rather than immediate consistency (though the user has control over the level of consistency, as will be explained) [3]. High availability and partition tolerance are achieved through asynchronous replication of rows over several nodes in the cluster, using consistent hashing and virtual nodes to handle high churn and incremental addition of nodes [16] [30] [31]. The amount of replicas can be chosen by the client. Furthermore, Cassandra provides cross-datacenter replication to cope with entire datacenter failures.
On reads and writes, the client can specify the desired quorum, that is the number of replicas that acknowledge the operation. Although Cassandra was built with eventual consistency in mind, strong consistency can be obtained by choosing the quorum larger than the number of replicas [22].
In terms of concurrency control, Cassandra supports atomicity for single-row operations and serializable *lightweight transactions*, essentially a compare-and-set functionality for larger operations [15].
Being an open-source project, Cassandra is freely available, but there is an enterprise version with extra features such as integration with the data processing engine Apache Spark and the distributed search platform Apache Solr, for complex analytical and search tasks [14] [12] [44] [19].

In conclusion, Cassandra offers flexible data-modeling with decent querying and indexing support through its CQL-interface and scales incrementally to vast datasets, thanks to its extensive replication and failure-handling features.

B. HBase

Apache HBase is an open source datastore using Google BigTable's datamodel, but running on top of the Hadoop Distributed File System (HDFS) instead of the Google File System (GFS).
Since its launch, HBase has adopted several secondary indexing mechanisms. For indexing, HBase also uses LSM trees [2] [38], and it also provides Bloom filters [21]. HBase comes with a Java API but without an SQL-like advanced querying language, though a workaround via Apache Hive, another data warehousing and analytics project [18], and its query language HiveQL is possible. Because of its HDFS underpinnings it can easily function as both input and output for MapReduce jobs.
HBase partitions data in ranges like BigTable and replicates updates in master-slave or multi-master fashion. Read-requests are not distributed however, as rows are only serviced by one server. The replicas are intended solely for failure recovery.
The strong points of HBase are its strong consistency, which is rare among NoSQL-stores, and its concurrency model: ACID-compliant single row transactions and optimistic multi-version concurrency control for wider scope operations [20] [22] [2].

In conclusion, HBase allows users to flexibly model data and is especially useful when there is already a HDFS dataset and when MapReduce compatibility is a priority. It scales well to very large datasets and features excellent concurrency control and strong consistency.

V. NewSQL

A. VoltDB

VoltDB is a relational in-memory distributed database, aiming to couple the guarantees of classical SQL-stores with the scaling of NoSQL systems while performing at very high speeds [40].
VoltDB stores data in the traditional relational model, but replicated and partitioned (using consistent hashing) over several nodes [22]. The data is queryable through a (growing) subset of SQL-92 [43]. Queries are preferably defined as stored procedures written in Java, in which the SQL statements are embedded. VoltDB supports primary and secondary indexing, and gives the user the choice between hash- and tree-indexes [41]. VoltDB plans and optimizes queries in stored procedures optimized at compile time [42].
Relying entirely on DRAM makes it expensive to scale to petabyte scale datavolumes, but VoltDB can export data to other, more suited DMBSs such as columnar NoSQL stores.
In terms of concurrency control, VoltDB supports full ACID-transactions which execute simultaneously on all replicas. Main memory is divided into chunks and these are statically assigned to individual, single-threaded cores. A global controller serializes all multi-node transactions to a sequence of single-node transactions and inserts those into the transactions queues of the respective nodes. In this way, VoltDB obviates the need for locking and latching techniques.

In short, VoltDB provides relational in memory storage for relatively large datasets, with very fast SQL-query capacities and ACID transactions. It is thus more suited for compute intensive applications that don't work on exorbitantly large data volumes but require very low latency.

B. Cloudera Impala

Cloudera Impala is a distributed SQL-engine running on top of the Hadoop stack, either on HDFS or HBase [29]. It is intended specifically for analytical use, focussing on delivering real-time querying capacities and not on high-throughput on writes.
Data is accessible through a subset of SQL-92.
Impala's architecture is almost perfectly symmetrically distributed: all nodes execute the same `impalad`-process, which is responsible for the main database functionality. Queries can be sent to any node which will then function as coordinator for the specific query. There are however two processes that run on only one (not necessarily the same) node in the cluster, performing bookkeeping tasks and relaying metadata changes through the cluster [6].
Contrary to the options discussed before, Impala doesn't partition rows by default. It does give the user the possibility to partition data should the data size call for it [7].

Impala has a highly efficient I/O-layer keeping disk- and CPU-utilization high at all times, resulting in considerably faster performance than other SQL-on-Hadoop solutions such as Apache Hive [17]. However, this comes with the drawback that the working set of a query has to fit in the aggregate physical memory of the cluster it runs on. This puts some restrictions on the size of datasets processable with Impala, not fully using the scaling capacity of the underlying Hadoop layer.

Because of its analytical purposes, Impala doesn't have extensive concurrency control features, relying instead on the underlying storage layer. With HBase's excellent concurrency control features, this doesn't necessarily pose a problem.

VI. Conclusions

In recent years, progress in microbiology and bioinformatics has driven down the cost of DNA-sequencing. As DNA of more and more organisms is being sequenced, the question arises to make this process ever more efficient and to store the generated data in a scalable, performant and accessible way. This study has focused on NoSQL and NewSQL datastores to be used in both the sequencing pipeline as in the storage and analysis of the results. These solutions offer elastic scalability, flexible data modelling, good behavior in a distributed setting and resilience to failures.

Specifically, this paper has reviewed 6 different datastores, 2 of the most popular ones in three categories, and compared them on a selection of properties relevant to HPC applications. Columnar stores like Apache Cassandra and HBase are well suited to the task of storing extremely large datasets in a reliable and performant way, and could handle the storage of already sequenced genomes. MongoDB sets itself apart with its extensive API, flexible data modeling, querying and indexing features. It doesn't have quite the write-handling capacities of the columnar stores, but is nevertheless a strong candidate for storing very large datasets such as the sequenced genomes. VoltDB and CouchBase Server offer low latency on reads and writes, albeit (especially VoltDB) on smaller datasets, but would thus be very useful for storing rapidly changing intermediate data in the sequencing pipeline. Lastly, Cloudera Impala lends itself well to situations where fast read, but not write queries on large datasets are required.

References

[1] Jason Baker, Chris Bond, James C Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Léon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, volume 11, pages 223–234, 2011.

[2] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, et al. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080. ACM, 2011.

[3] Eric A Brewer. Towards robust distributed systems. In *PODC*, page 7, 2000.

[4] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[6] Cloudera. Cloudera impala components. http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_components.html. Accessed on 6/12/2014.

[7] Cloudera. Cloudera impala partitioning. http://www.cloudera.com/content/cloudera/en/documentation/cloudera-impala/v1/latest/Installing-and-Using-Impala/ciiu_partitioning.html. Accessed on 03/12/2014.

[8] Pascal Costanza. elprep-mongo. https://github.com/ExaScience/elprep-mongo. Accessed on 25/11/2014.

[9] Couchbase. About couchbase server. http://www.couchbase.com/nosql-databases/about-couchbase-server. Accessed on 16/11/2014.

[10] Couchbase. Couchbase compaction process. http://blog.couchbase.com/compaction-magic-couchbase-server-20. Accessed on 26/11/2014.

[11] Couchbase. Couchbase n1ql language reference. http://docs.couchbase.com/developer/n1ql-dp4/n1ql-intro.html. Accessed on 6/04/2014.

[12] DataStax. Analyzing data using spark. http://www.datastax.com/documentation/datastax_enterprise/4.5/datastax_enterprise/spark/sparkTOC.html. Accessed on 15/11/2014.

[13] DataStax. Cassandra query language (cql) v3.1.7. http://cassandra.apache.org/doc/cql3/CQL.html. Accessed on 15/11/2014.

[14] DataStax. Getting started with solr in datastax enterprise. http://www.datastax.com/documentation/datastax_enterprise/4.5/datastax_enterprise/srch/srchIntro.html. Accessed on 15/11/2014.

[15] DataStax. Lightweight transactions. http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_ltwt_transaction_c.html. Accessed on 15/11/2014.

[16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[17] Avrilia Floratou, Umar Farooq Minhas, and Fatma Ozcan. Sql-on-hadoop: Full circle back to shared-nothing database architectures. *Proceedings of the VLDB Endowment*, 7(12), 2014.

[18] Apache Foundation. Apache hive. https://hive.apache.org/. Accessed on 06/12/2014.

[19] Apache Foundation. Apache solr. http://lucene.apache.org/solr/. Accessed on 02/12/2014.

[20] Apache Foundation. Hbase acid semantics. http://hbase.apache.org/acid-semantics.html. Accessed on 15/11/2014.

[21] Apache Foundation. Hbase schema design. http://hbase.apache.org/book/perf.schema.html. Accessed on 26/11/2014.

[22] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam AM Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):22, 2013.

[23] Robin Hecht and S Jablonski. Nosql evaluation. In *International Conference on Cloud and Service Computing*, 2011.

[24] Charlotte Herzeel, Pascal Costanza, Wolfgang De Meuter, and Thomas J. Ashby. Resolving load balancing issues in bwa on numa multicore architectures. In *Proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics PPAM*. Springer, 2013.

[25] imec. Exascience life lab. http://www2.imec.be/be_en/research/life-sciences/exascience-life-lab.html. Accessed on 02/12/2014.

[26] Broad Institute. Coverage. https://www.broadinstitute.org/crd/wiki/index.php/Read_coverage. Accessed on 05/12/2014.

[27] Broad Institute. Human whole genome sequencing. http://www.broadinstitute.org/scientific-community/science/platforms/genomics/human-whole-genome-sequencing. Accessed on 02/12/2014.

[28] Wetterstrand KA. Dna sequencing costs: Data from the nhgri genome sequencing program (gsp). http://www.genome.gov/sequencingcosts/. Accessed on 19/11/2014.

[29] Marcel Kornacker and Justin Erickson. Cloudera impala. http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/. Accessed on 15/11/2014.

[30] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[31] Avinash Lakshman, Prashant Malik, and Jonathan Ellis. Facebook's cassandra paper, annotated and compared to apache cassandra 2.0. http://www.datastax.com/documentation/articles/cassandra/cassandrathenandnow.html. Accessed on 15/11/2014.

[32] John R Mashey. Big data and the next wave of infrastress. In *Computer Science Division Seminar, University of California, Berkeley*, 1997.
[33] MongoDB. Mongodb gridfs. http://docs.mongodb.org/manual/core/gridfs/. Accessed on 6/12/2014.
[34] MongoDB. Mongodb indexes. http://docs.mongodb.org/manual/core/indexes-introduction/. Accessed on 15/11/2014.
[35] MongoDB. Mongodb query plans. http://docs.mongodb.org/manual/core/query-plans/. Accessed on 16/11/2014.
[36] James K Mullin. A second look at bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.
[37] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.
[38] Russell Sears and Raghu Ramakrishnan. blsm: a general purpose log structured merge tree. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 217–228. ACM, 2012.
[39] solid IT. Db-engine ranking. http://www.db-engines.com/en/ranking. Accessed on 15/11/2014.
[40] Michael Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.
[41] VoltDB. Voltdb indexes. http://mockdocs.voltdb.com/UsingVoltDB/ddlref_createindex.php. Accessed on 8/12/2014.
[42] VoltDB. Voltdb query plans. http://docs.voltdb.com/PerfGuide/ChapExecPlans.php. Accessed on 8/12/2014.
[43] VoltDB. Voltdb technical overview. *Whitepaper*, 2010.
[44] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.