2019 IEEE 35th International Conference on Data Engineering (ICDE)

AsterixDB Mid-Flight: A Case Study in Building Systems in Academia

Michael J. Carey[*]
Computer Science Department
University of California-Irvine
Email: mjcarey@ics.uci.edu
([*]Also affiliated with Couchbase, Inc.)

***Abstract*—Building large software systems is always a challenging venture, but it is especially so in academia. This paper describes the experiences that the author and his (mostly UC-based) partners in software crime have had that culminated in the Big Data Management System now available as Apache AsterixDB. It covers a mix of the history and technical content of the nearly ten-year-old project, starting with its inception during the MapReduce craze. It describes the phases that the effort has gone through and some of the lessons learned along the way. The paper also covers some personal reflections and opinions about the challenges of systems-building, as well as writing about it, in our current academic culture. Included is the case for doing this sort of work at all – discussing the pitfalls of doing "systems" research in the absence of an actual system, and why the gain outweighs the pain of building and sharing database software in academia. As of late 2018, Apache AsterixDB is also having a commercial impact as the storage and parallel query engine underlying a new offering called Couchbase Analytics. The last part of the paper explains how we are attempting to balance the uses of AsterixDB as (i) a generally available open source Apache software platform, (ii) an end-to-end research testbed for universities, and (iii) the technology powering a commercial NoSQL product.**

***Keywords*-Big Data; software development; academia;**

I. Introduction

Building large software systems is a challenging venture. This is especially true in an academic setting. In academia, the vast majority of the software team members (students) come and go more frequently than they do in industry. Moreover, the reward system in academia – with its focus on conference and journal papers and peer-reviewed research grants – is not always as "software project friendly" as one might wish it to be. The purpose of this paper is to examine the AsterixDB system – which is now available to all as Apache AsterixDB – as one potentially interesting and informative case study in academic software building. Included in the examination will be how and why the project got started, how it was initially received (and perceived), and what some of the challenges and lessons have been in terms of its execution, funding, and paper-writing activities. Readers are warned that, due to its nature, this paper will be heavy on self-citations and light on related work. (Other systems are welcome to tell their own stories!)

The remainder of this paper will be organized as follows: Sections II and III set the stage by providing a bit of background about the author followed by a technical overview of the Apache AsterixDB software platform itself. Section IV describes the history of the AsterixDB project and some of the challenges and lessons learned. Section V is a more general discussion of reasons to build large software systems in academia despite the challenges that one faces in doing so. Section VI provides a brief description of how AsterixDB is now being used in one commercial setting, and Section VII explains the resulting balancing act that we are dealing with on an ongoing basis as a result of our desires for the system. Section VIII concludes the paper.

II. A Bit of Personal History

Before proceeding, it may be useful for the reader to know a bit about the author's pre-AsterixDB academic background and resulting mindset. I was extremely fortunate to have spent five of my formative years as an Electrical Engineering B.S. and subsequently M.S. student, one with an interest in computing – but before there was an actual Computer Science major – at Carnegie-Mellon University. Upon getting off the elevator on the 3rd floor of Science Hall, one was immediately faced with an inspiring collection of flashing red LEDs: a glassed-in machine room that held two of the world's earliest multiprocessor computer systems, namely C.mmp (a 16-processor shared-memory computer system [1]) and Cm* (a 50-processor NUMA system [2]). A favorite professor and mentor, and later M.S. thesis co-supervisor, was Anita Jones, leader of the StarOS [3] operating system project for Cm*. I also met a finishing Ph.D. student and soon-to-be Berkeley professor, John Ousterhout, leader of the competing Medusa [4] operating system project for Cm*. Parallel computing and systems building were in the air at CMU, and I was thoroughly hooked by the time I left to pursue a Computer Science Ph.D. at UC Berkeley.[1]

[1]I declined an offer from Cornell after a now-senior faculty member there advised me, during a visit, that systems-building was not an appropriate part of one's Ph.D. work, even in "systems." One's Ph.D. years were a time for theory and principles, not programming.

At Berkeley, I found myself at the right place at the right time from an academic systems-building perspective. My initial academic advisor was Dave Patterson, co-recipient of the 2017 ACM Turing Award for his work on RISC architectures and quantitatively-driven architecture design and evaluation – work that was starting at that time. Also joining the fray there was the aforementioned John Ousterhout. Last in this tale, but arguably not least, was a recently tenured database professor, Mike Stonebraker. Anita Jones at CMU had given me strict orders to check out his courses and his INGRES database project once I got to Berkeley. Although "databases" sounded (seriously) boring, I followed orders, and hence discovered another group that was building a large system in academia and having a big impact on the database system revolution happening at that formative time [5].

After graduating from Berkeley I moved to a faculty position at the University of Wisconsin. My faculty mentor there was David DeWitt, and his now famous Gamma project [6] on shared-nothing parallel database systems was just getting underway. David soon invited me to co-define/co-lead a new project on extensible database systems that he was wanting to start – EXODUS [7] – and later, together with colleagues Jeff Naughton and Marvin Solomon, we ran the Shore project on object-oriented database systems [8].

III. AsterixDB System Overview

To fully appreciate AsterixDB as a case study in large academic software, the reader should probably know what AsterixDB actually is. This section covers the prerequisite *what*; later sections will cover the *why* and *when*, the *how*, the tradeoffs, and other aspects of the overall adventure.

Apache AsterixDB [9] is a Big Data Management System (BDMS) with a feature set chosen to target use cases such as web data warehousing and social media data analysis. Its notable features include:

1) A NoSQL-style data model (ADM) based on extending JSON with object database concepts;
2) Two declarative query languages (first AQL and later SQL++) that support a broad range of queries against multiple semi-structured datasets;
3) A rule-based, data-partition-aware query optimizer for parallel queries (Algebricks);
4) An efficient dataflow execution engine (Hyracks) for partitioned-parallel execution of query plans;
5) Partitioned and LSM-based native storage and indexing for large datasets;
6) Support for querying and indexing of external data (e.g., data in HDFS) as well as natively stored data;
7) Rich data type support, including numeric, textual, temporal, and simple spatial data;
8) Secondary indexing through B+ trees, R-trees, and several variants of inverted keyword indexes;
9) Basic NoSQL-like transactional capabilities similar to those of popular NoSQL stores.

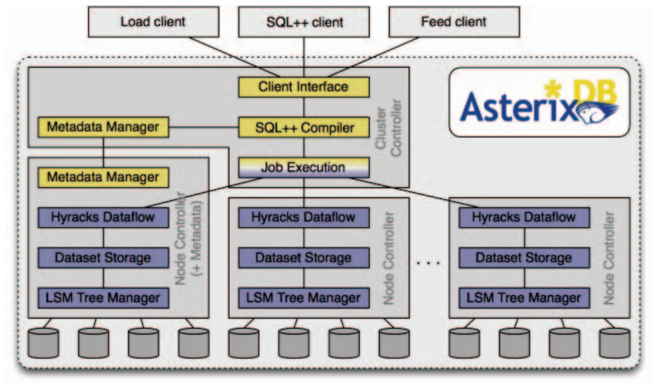

Figure 1. AsterixDB system schematic

Figure 1 shows a high-level overview of AsterixDB. The system is based on a traditional shared-nothing architecture, with each node in a cluster managing one or more storage and index partitions for its datasets based on LSM (Log-Structured Merge) tree technology. Each node in the cluster runs an instance of the Hyracks dataflow platform for query execution, and the execution of the Hyracks jobs resulting from SQL++ query requests is coordinated by the cluster controller. Targeting truly "Big Data" of the sort expected in the initial web and social media analytics use cases, a fundamental assumption from the start of the project has been that the portion of data stored on a given node can well exceed the size of its main memory, and likewise (at least potentially) for intermediate query results [10].

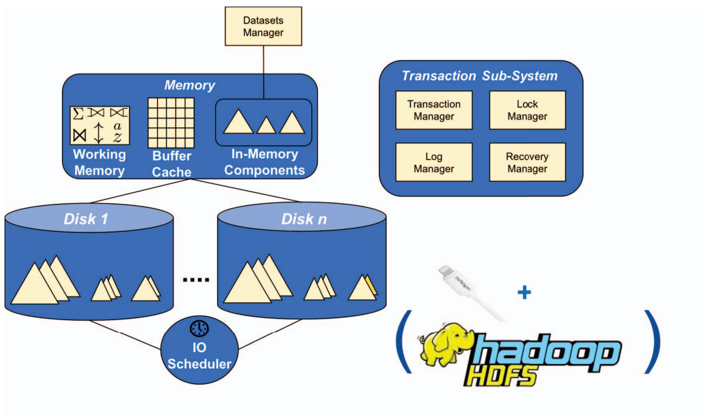

Figure 2. AsterixDB storage and memory management

Figure 2 zooms in a bit more closely on the storage and resource details of a node in an AsterixDB cluster. Each node can have multiple I/O devices, with each storing the LSM components associated with a dataset partition. Each node uses its memory for a mix of ingestion buffering (via the LSM memory components of active datasets), buffering of pages of LSM disk components as they are accessed (via the buffer cache), and processing of memory-intensive operations including sorts, joins, and grouped aggregation

```
CREATE TYPE GleambookUserType AS {
    id: int,
    alias: string,
    name: string,
    userSince: datetime,
    friendIds: {{ int }},
    employment: [EmploymentType]
};

CREATE TYPE GleambookMessageType AS {
    messageId: int,
    authorId: int,
    inResponseTo: int?,
    senderLocation: point?,
    message: string
};

CREATE TYPE EmploymentType AS {
    organizationName: string,
    startDate: date,
    endDate: date?
};

CREATE DATASET GleambookUsers(GleambookUserType)
PRIMARY KEY id;

CREATE DATASET GleambookMessages(GleambookMessageType)
PRIMARY KEY messageId;

CREATE INDEX gbUserSinceIdx ON GleambookUsers(userSince);

CREATE INDEX gbAuthorIdx ON GleambookMessages(authorId)
    TYPE BTREE;

CREATE INDEX gbSenderLocIndex ON
              GleambookMessages(senderLocation)
    TYPE RTREE;

CREATE INDEX gbMessageIdx ONGleambookMessages(message)
    TYPE KEYWORD;
```

(a)

```
CREATE TYPE AccessLogType AS CLOSED {
    ip: string,
    time: string,
    user: string,
    verb: string,
    `path`: string,
    stat: int32,
    size: int32
};

CREATE EXTERNAL DATASET AccessLog(AccessLogType)
USING localfs
(("path"="localhost:///Users/mjc/extdemo/accesses.txt"),
 ("format"="delimited-text"), ("delimiter"="|"));
```

(b)

```
WITH endTime AS current_datetime(),
     startTime AS endTime - duration("P30D")
SELECT nf AS numFriends, COUNT(user) AS activeUsers
FROM GleambookUsers user
LET nf = COLL_COUNT(user.friendIds)
WHERE SOME logrec IN AccessLog SATISFIES
          user.alias = logrec.user
  AND datetime(logrec.time) >=
startTime
  AND datetime(logrec.time) <=
endTime
GROUP BY nf;
```

(c)

```
UPSERT INTO GleambookUsers (
{   "id":667,
    "alias":dfrump",
    "name":"DonaldFrump",
    "nickname":"Frumpkin",
    "userSince":datetime("2017-01-01T00:00:00"),
    "friendIds":{{ }},
    "employment":[{"organizationName":"USA",
    "startDate":date("2017-01-20")}],
    "gender":"M"}
);
```

(d)

Figure 3. Examples of (a) ADM data types, datasets, and indexes, (b) an external dataset definition, (c) a SQL++ SELECT query, and (d) a SQL++ UPSERT statement

(using the working memory). The figure also alludes to the fact that data in HDFS files can be made accessible for querying in situ (vs. being imported) in AsterixDB.

Figure 3 illustrates the user model of AsterixDB. It shows the AsterixDB data type, dataset, index, and external dataset definitions for a hypothetical social media web site as well as showing a SQL++ `SELECT` query and a SQL++ `UPSERT` statement. AsterixDB users can define ADM data types specifying whatever schema information is known a priori; instances of ADM types are also allowed by default to carry additional (self-describing) record content. The provision of schema information is optional, so it is entirely up to the definer of an application to choose what (and how much, if any) to predeclare. If desired, a type can be marked "closed" to forbid the inclusion of extra fields in its object instances.

The `GleamBookUserType` in Figure 3 shows a few of the basic data types supported by ADM and how one can compose complex object types using a mix of primitive types and record, list, and unordered list (multiset) constructors. Fields can be declared as optional (notice the "?" fields in the figure) or they can be omitted altogether from the (partial) schema for a given type. (E.g., one could build the same example application to manage the same data without declaring anything but the id field for the data type GleamBookUserType.) ADM thus enables the developers of an application to choose an essentially schema-free world, a highly-specified schema world, or something in between when developing their AsterixDB application. Part (b) of Figure 3 shows how one can make external data such as a log file queryable as if it were natively stored using AsterixDB's external dataset support. (It also illustrates a use of a closed data type.) The query in part (c) illustrates the use of SQL++ for the analysis of such data; it examines a mix of stored and external data in order to compute a summary of the number of recently active users of the social website grouped by their number of friends. The statement in part (d) illustrates a simple update in SQL++; the example either inserts a new GleamBook user with id 667 or replaces the one that's currently there (if such a user already exists).

AsterixDB's data storage scales linearly through primary key-based hash partitioning of all datasets. The data objects in a given dataset are stored in partitions of LSM-based B+

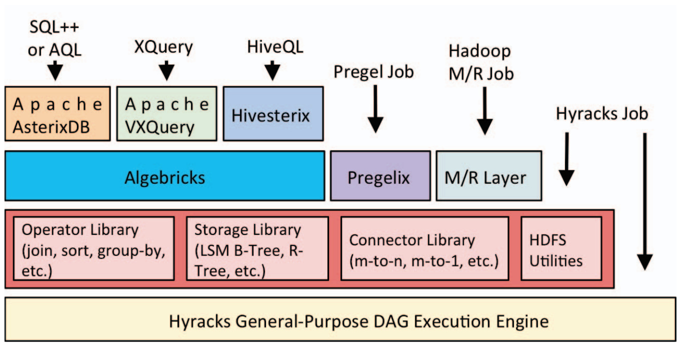
SQL++ or AQL
XQuery
HiveQL
Pregel Job
Hadoop M/R Job
Hyracks Job
Apache AsterixDB
Apache VXQuery
Hivesterix
Algebricks
Pregelix
M/R Layer
Operator Library (join, sort, group-by, etc.)
Storage Library (LSM B-Tree, R-Tree, etc.)
Connector Library (m-to-n, m-to-1, etc.)
HDFS Utilities
Hyracks General-Purpose DAG Execution Engine

Figure 4. AsterixDB software stack

trees, and local secondary indexing of the data partitions can be requested by creating any combination of B+ trees, R-trees, and inverted indexes [11]. The runtime engine for executing SQL++ queries in AsterixDB is the Hyracks data-parallel platform [12], an extensible dataflow platform that at one point was scale-tested on a large (180 nodes and 1440 cores) cluster [13] that Yahoo! Research gave us access to.

Figure 4 illustrates the layered software stack underlying AsterixDB. It shows the major layers that make up the system as well as various other uses (shown across the top of the figure) that have been made of the system's designed-for-reuse components. As one example, the Apache VXQuery project also makes use both of Hyracks and of the stack's Algebricks algebraic planning and optimization framework [14]. Other uses that have been made of the stack over the years include an implementation of HiveQL using all layers and Hyracks-based versions of a Pregel-like graph analytics engine and a (slightly generalized) Hadoop MapReduce engine clone. For AsterixDB, its SQL++ queries are compiled and optimized through a combination of AsterixDB-specific code plus a significant body of shared rules and code provided by the Algebricks extensible algebraic parallel query planning and optimization framework, which is shown in Figure 5.

IV. History, Challenges, and Experiences

Figure 6 shows a timeline of the AsterixDB adventure to date. The idea for the project first formed in 2008 when the author reentered academia after a 13-year hiatus in industry. Given my long-time interests in parallel computing, parallel databases, and systems building, my UCI colleague Chen Li and I started with the question "What sort of system might we want to build today if we had a cluster?" At that time, XML and XML databases were still in vogue, and it was the dawn of the Hadoop era – HDFS and MapReduce were rapidly gaining popularity both in industry and (because of that) with systems-oriented database researchers. To learn about MapReduce, given Chen's similarity search interests

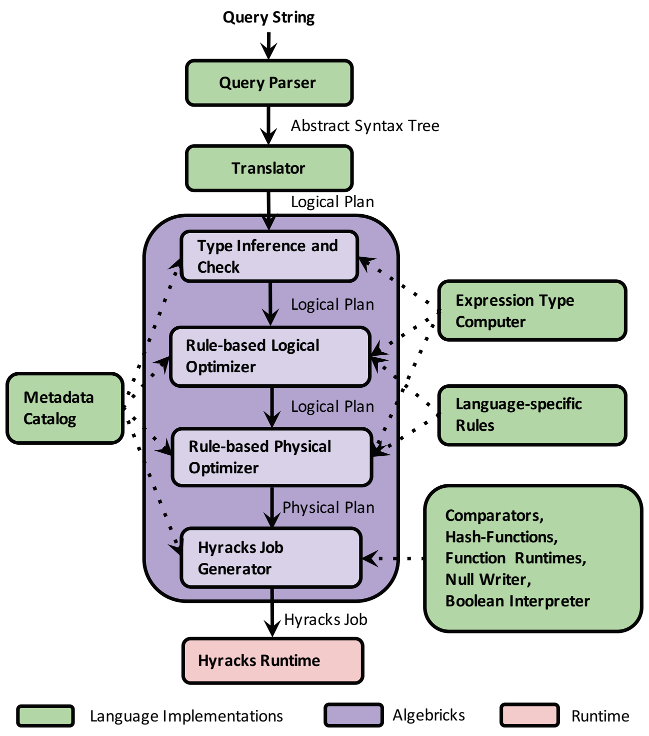

Figure 5. Algebricks query compiler framework

and expertise, we convinced one of Chen's Ph.D. students to work on parallel similarity search on that platform (see [15] for the eventual result). It became clear to us pretty quickly that MapReduce was *not* a sensible runtime platform for efficient, database-style query processing, so we decided to answer our cluster question by proposing a project to NSF to build a new scalable, open source, parallel database system for XML – called ***ASTERIX***, initially short for ***A****ctive*, ***S****calable*, ***T****ransactional* ***E****nterprise* ***R****epository* for ***I****nformation in* ***X****ML*. As mentioned earlier, the kinds of use cases we had in mind were tasks like warehousing and analyzing web data, social media data, message data, and other soon-to-be voluminous data – data whose objects are richer and more diverse and complex than relational data, data that has textual, temporal, and simple (Google-map style) spatial attributes as well as the more traditional numeric- and string-valued fields.

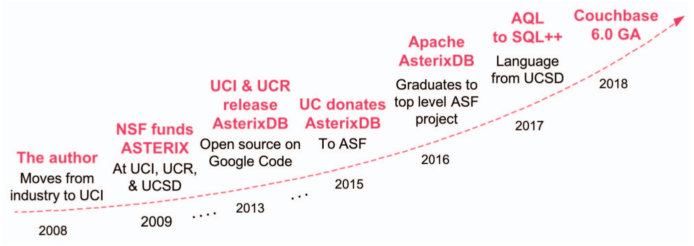

Figure 6. AsterixDB Project Timeline

A. In The Beginning

In order to maximize our chances of success, as we were about to approach NSF with a multi-$M funding request for the ASTERIX project, we formed a scalable XML "dream team" by drawing faculty from three UC campuses in southern California: Chen Li and I from UC Irvine, Vassilis Tsotras from UC Riverside, and Yannis Papakonstantinou and Alin Deutsch from UC San Diego. We received the funding and commenced work in earnest in 2009. One of the big things we did to help ensure that we would build a solid software system was to hire a professional software architect and developer (Vinayak Borkar) with whom the author had worked for a number of years at BEA systems. We also subsequently hired a full-time postdoc (Nick Onose) from UC San Diego. This team, along with a few "founding students" at UCI, fleshed out the initial ASTERIX software vision [16].

Logistically, designing a large software system with a distributed (even locally) development team proved to be more challenging than we expected. As the project unfolded, UCI and UCR were able to coordinate closely because UCR had their students spend a day a week at UCI to participate in person in project meetings. Fairly early on in the project, we realized that XML had peaked and that JSON was the "new kid" on the semistructured data block. A decision was made to base the ASTERIX data model on extending JSON with modeling features that one would want in a database-oriented data model – e.g., support for multisets in addition to lists, more data types, and the aforementioned open type system and its schema language. The first query language was designed by borrowing many good ideas from the W3C XML query language, XQuery, as XQuery was well-suited for semistructured, schemaless data; the XML cruft in XQuery (features related to document order, mixed content, XPath, ...) was thrown overboard.

As shown in the timeline, the initial NSF project and development effort ran from mid-2009 to mid-2013. During that time, UCI and UCR worked closely on the new ADM/AQL track and were responsible for the software development; UCSD stayed closer to the original XML-oriented track, independently doing research related to large-scale document data and search. As promised in the initial grant proposal, 2013 saw the release of a first complete version of the system – as AsterixDB – in open source via Google Code. Over the course of the early years, the UCI and UCR team leads and students made pilgrimages to visit companies like eBay, Facebook, Google, Yahoo!, Teradata, HTC, Netflix, LinkedIn, and Twitter to show them our work in progress and solicit critical feedback and suggestions.

Towards the latter part of the first phase, we approached a handful of companies about potentially sponsoring the ASTERIX effort via industrial gifts. One of the respondents, Oracle, gave us the gift of professional manpower – they offered to hire a new Oracle Labs employee whose focus for roughly two years would be on working with us full-time on AsterixDB. Another ex-BEA team member (Till Westmann) joined the team at Oracle Labs as a result and became a remote senior contributor to the ongoing effort.

The second phase of the project ran from 2013-2017, and was funded by NSF's software infrastructure program supplemented by industrial gift funds. During this phase, the funded focus of the AsterixDB project was on hardening the code base, creating more documentation, studying and improving the platform's performance, starting to cultivate and support a modest user base, and adding a number of non-research features needed for the system to be more usable in practice. Towards the grow-a-user-base goal, we approached UCI and UCR about waiving UC IP concerns and donating the project to Apache, as Apache was the home for most of the successful open source Big Data infrastructure projects. We cited our original NSF grant, in which we had promised unrestricted sharing of the results and code base, and cited the example of UC Berkeley having recently allowed Spark to do that. They agreed, and they handed UC's ownership of the system back to all of its contributors; each of us then signed our rights over to the ASF. That donation landed AsterixDB in the Apache incubator for a year or so, and the project successfully graduated to top-level project status in 2016. During that period we also launched a new NSF research project on "Big Active Data" (BAD) that led to an extension of AsterixDB with features that might be roughly characterized as "data pub/sub" [17].

There have been several significant events on the AsterixDB timeline since 2016. One is that, due to an introduction made by a mutual friend, discussions began with Couchbase, Inc. Couchbase is a NoSQL software vendor that had a scalable, JSON-based document store with a SQL-like query language (albeit a slightly quirky one in the early days). Section VI will have more to say about that. The other significant event was the addition of, and eventual switch to, a new query language, SQL++ [18].

Prior to SQL++, as mentioned earlier, we had designed and implemented our AQL language. Recall that AQL came from taking XQuery, which was designed by a group of experts for querying semistructured XML data, and tossing out its XML cruft in order to create a query language for JSON and ADM. To make AQL friendlier to SQL users, we later added support for some SQL-inspired keyword substitutions (e.g., `SELECT` for `RETURN`, `FROM` for `FOR`). We had a reasonable technical explanation about why AQL was what it was – i.e., why it wasn't just a SQL superset. Our users politely listened to our story and learned AQL, but they kept wishing that AQL was more like SQL (in syntax and not just query power). In parallel, Yannis Papakonstantinou had a team at UCSD working on a new project called FORWARD, a data integration system for integrating data from heterogeneous data stores. SQL++ was a SQL extension for querying semistructured data in FORWARD. Yannis approached us with a draft of their SQL++ design [18], first getting our attention with news that AQL had fared well in their comparison of query languages for JSON. On further study, we saw that SQL++ was very much like AQL, but with a SQL-based syntax that would make AsterixDB users much happier. SQL++ did a nice job of mostly extending standard SQL, while allowing for differences in a few key places where SQL made flat-world or schema-based assumptions that no longer hold for ADM or JSON, and of exploiting the nested/composable data model of JSON by offering generalized support for grouping and aggregation [19]. Thanks to AsterixDB's Algebricks and Hyracks layers, we were able implement SQL++ fairly quickly as a peer of AQL, sharing the Algebricks query algebra and many optimizer rules as well as the associated Hyracks runtime operators and connectors. We have now deprecated AQL in favor of SQL++.

B. Funding and Publishing

While AsterixDB has enjoyed a series of NSF grants over the years, for which we are grateful, we have definitely had more proposal failures than grant successes along the way. In general, we have found that some reviewers in our field, at least in this author's opinion, have a "bad attitude" when it comes to research that aims to build relatively complete systems and/or to measure them. We have gotten used to reviewer feedback such as "where's the innovation?" or "this is just engineering!" or "they should really just start a company to do this." (This is discouraging when compared with earlier days in the CS field, e.g., the era discussed in Section II.) Section V will tackle the question of some of the ways in which this is unhealthy for our field.

In terms of what proposal reviewers seem to like and dislike, it seems that "innovation" is strongly favored over "incremental work" or "engineering" by reviewers today. Unfortunately, as Section V will discuss further, this is in many ways the opposite of what is needed for most research results to have an actual impact. Significant improvements to data structures or algorithms that are actually in use in practice can be – as should be obvious – significant. In contrast, complex and incomplete new approaches that have not been tried before – perhaps for good reason – seem to be preferred by reviewers. Unfortunately, such work has less hope of ever escaping the pages of the papers it is eventually published in, or the CVs where those papers are later listed, and of having an actual impact.

We have also found – due to the fact that AsterixDB has in many ways been a counter-cultural effort – that proposal reviewers have a "bandwagon problem." As discussed earlier, AsterixDB was conceived in the early Hadoop era, a time when MapReduce was all the rage. Everyone seemed to be jumping on the Hadoop bandwagon – handing data management over to the distributed file system community, and using a 2-3 instruction runtime system (*map*, *reduce*, and optionally *combine*) to run queries by scanning data – and perhaps translating (Pig, Hive, ...) queries into MapReduce programs to run over HDFS files. As long-time database folks, that seemed very wrong to us, so we made our decision to skip that generation of work and instead build a platform where Big Data was handled by extending the field's three decades of research on parallel database systems as well as its decade or so of research on nested (NF^2) and semistructured (e.g., XML) query languages [20]. To illustrate the bandwagon problem, here is a snippet from an actual review that we received for a research proposal call that we had responded to:

"This is a very ambitious proposal, would strongly encourage trying to build atop Hadoop, even if its current filesystem and MR engine are not ideal for addressing this kind of problem. The MR engine is a layer on top (one that Pig uses behind the scenes), while the Distributed File System APIs and HDFS implementation are broadly useful. Work done on top of Hadoop is more likely to be picked up and integrate with other applications."

Another bandwagon issue in our field was created by the "one size no longer fits all" series of papers, e.g., [21]. I sometimes joke that the number of "sizes" needed to "fit all" today is correlated to the number of database startup companies in the Boston area. AsterixDB is definitely counter-cultural today in taking a more complete system, *a.k.a.,* "one size fits a bunch", approach to data management. This going-against-the-flow approach has hurt us with proposal reviewers on occasion, and it has also caused our project – which owns its data – to take longer to mature than, say, efforts like Spark or Flink that didn't try to tackle as large an overall problem from the outset.

One last funding-related point worth making here relates to project staffing for large systems projects. Wisconsin projects like EXODUS and Shore happened at a time when DARPA was investing in data management in a bigger way. We enjoyed multi-$M budgets and were able to hire multiple, high-quality, MS-level programming staff members to work on those projects along with our faculty and graduate student team. It is much more difficult to obtain that level of funding today, and in addition, industrial salaries have skyrocketed. Full-time staff can be critical to making a large systems project happen – yet they are very difficult to fund anymore. (We currently have one lone staff member on the UCI/UCR side of the Apache AsterixDB effort.)

Turning to our experiences in publishing our work on AsterixDB, we have, not surprisingly, encountered similar "attitude issues" in paper reviews. Again, reviewer feedback like "where's the innovation?" or "this is just engineering!" is not an uncommon reaction to systems papers; this is a fairly common experience for systems builders in our field. Jeff Naughton gave what is probably my favorite diatribe on this topic (our current publication culture and its dangers) as kind of a "surprise" keynote (topically) at ICDE in 2010 [22]. Two particular challenges that we have encountered in our AsterixDB publishing adventures, and found frustrating, are:

1) *Damned if you do, damned if you don't!* If you work in the context of a system like AsterixDB, some reviewers will accuse you of producing system-specific results and seek to reject the work because it is in the context of one particular system. Despite the system providing a common ground for evaluating a set of alternative approaches (see Section V), such a reviewer will object to the work's software context. (Substitute your favorite system for AsterixDB.) This is a legitimate concern, but if the results' broader applicability are explained, this should suffice to offset such criticism. Some reviewers seem not to be open to considering such generality arguments.
2) *How hard can that be?* Some reviewers have no clue about how much work, and how many details, must be dealt with to work in the context of a real system. As an example, one of our PhD students did a comparison of LSM-based versions of various spatial access methods that eventually appeared as [23]. However, the work was previously submitted, reviewed, and invited for revision for another conference. One of those reviewers suggested that, during the 1-3 month revision window, we should implement two more access methods (on top of 4-5 already in the paper) and run all of our experiments on them as well. Those two access methods have never been used outside of papers and simulations because the details required to load them, make them recoverable, and make them concurrent have never been figured out. Those are prerequisites to adding a new access method in AsterixDB, as the system must work end-to-end, and the costs of such mechanisms must be included in any useful study. Our student was accused of "lack of effort" for not following that suggestion on top of the rest of his revision work, and the reviewer saw to it that the paper was soundly rejected after revision.

For better or for worse, in terms of the length of our graduating students' CVs, we have adopted what we sometimes refer to as the "BMW model" for publishing in AsterixDB. **BMW** stands for **B**uild, **M**easure, **W**rite. Our project philosophy is that the goal is to solve interesting problems and build "cool stuff" – not to write papers. Papers are for communicating results – they are for sharing the fruits of our labor – they are not themselves the primary objective of the work that they report on. Thus, we require our students to first build things, and then evaluate those things, and only then pop their heads up to look around for venues that might be appropriate targets for sharing their results.

Along those lines, two "software heroes" whose work and impact I have always found especially admirable are: (1) Barbara Liskov, who has had a remarkably impactful career at MIT – involving contributions such as the CLU programming language [24] and Argus distributed transaction system [25] – and won a much-deserved ACM Turing Award. (2) John Ousterhout, mentioned in Section II, who has "done time" in both academia and industry over the years, and contributed to diverse areas including networked operating systems, VLSI design tools, scripting languages, log-structured file systems, high-performance distributed systems, and more. Readers are strongly encouraged to go to their favorite publication-related web site (e.g., DBLP or Google Scholar) and observe the number of papers per year that those two individuals have published – then contrast their impact with those of other "highly-published" individuals with much longer CVs. There's an important lesson to be learned, in my opinion. (One simply cannot do more than a few truly paper-worthy things per year.)

V. Why Build Systems in Academia?

Despite the challenges of doing systems research in academia, I believe that it is well worth doing. In part it's worth doing if you are "called to do it" – i.e., if you find it rewarding to envision things that don't yet exist, build them, and then see them actually work (and hopefully be used by others). However, it's also worth doing because doing it has a number of benefits, helping to guide your work away from things that aren't worth doing and towards things that are. Reasons include...

A. *To Make Sure It's Possible*

For systems researchers, we have a variety of tools at our disposal. We can build and measure systems, we can build mock-ups of pieces of systems, or we can design and maybe simulate them and hope we're not missing something. The build-and-measure approach can be critical to not taking an accidental wrong turn.

As an example, in the late 1980's and early 1990's, an interesting (perhaps ahead of its time) sub-area of database systems research was "multidatabase systems." This was a federated data, or virtual data integration, thought. A set of otherwise autonomous heterogeneous database systems could be coupled together and presented to users as a single-system image. Some groups built things; others studied the problems and wrote papers without doing so. One sub-problem in building a multidatabase system is: What about transactions? Some of the "paper groups" studied the problem, designing system-level concurrency control schemes that could tolerate the coupling of multiple systems running different concurrency control ideas from the literature – locking, optimistic, timestamps – and many papers were written. Unfortunately, a number of those papers got two things wrong. First, most real database systems used locking – as many of the practical problems (e.g., index concurrency control, crash recovery, ...) had not been solved for non-lock-based methods. Thus, the degree of assumed heterogeneity was off – so this was arguably a non-problem, at least at the time. Second, many of the published algorithms assumed that the systems would report readset and writeset information back to a distributed transaction manager. Unfortunately, database systems just don't work that way. Their actual APIs speak SQL, referring to tables and predicates, not read(x) and write(x) actions. Moreover, the item granularity (records, pages, files, index pages) varies from system to system. Doing their work within the context of a real system, or at least on top of some real systems, could have helped to guide those research efforts in more applicable and potentially impactful directions.

B. To Make Sure It's Beneficial

Systems researchers are engineers, and systems are made up of components. Some of them are extremely interesting, and it's easy to want to deep-dive into their designs – possibly without checking to see if doing so will be beneficial.

An example here is the backstory for the previously mentioned LSM spatial data publication, namely [23]. That work was inspired by a "perfect storm" of comments that we got within a relatively short period of time a few years ago. For spatial data, AsterixDB was offering R-tree based indexing. In response to our sharing that information in talks, several respected senior database researchers asserted that everyone knows that X is the best way to index spatial data – but for different values of X. One was essentially patting us on the back for having made the wise choice of LSM-based R-trees. Another was criticizing us for not having gone with the approach of linearizing 2D data (e.g., via Hilbert-ordering or Z-ordering) and using LSM-based B-trees on the transformed spatial keys. A third argued in a visit to UCI that a grid-based approach would probably be better. Add to that a musician friend who daylights as a Microsoft SQL Server app developer – he asked for help understanding and tuning a spatial index for a SQL Server application he was building. We became curious about who was right, and what the tradeoffs would be, and it seemed that not much had been done in "LSM land" for spatial data.

We undertook the study, but the end results were – to one of my collaborators – a bit "disappointing." Our PhD student implemented a variety of different methods, getting them to work end-to-end, and he then ran performance tests on them by running spatial queries (end-to-end queries, in AQL at the time) against a cluster running AsterixDB while varying the data sizes and other query and data characteristics. Despite trying some very different structures, the performance differences were relatively minor [23]. On digging deeper, it turned out that the performance bottleneck in the queries was elsewhere – once the spatial index had identified qualifying objects, the objects themselves had to be fetched in order to answer the queries in an end-to-end sense. Although AsterixDB employs the usual tricks to speed up indexed data access (e.g., sorting object references, which in our case are primary keys, before fetching data objects [26]), the spatial index portions of the overall query times were sufficiently small to make differences between index types noticable but relatively minor – that is, though some of the differences between them *within* their portion of the query times were significant, those index time differences were watered down to the ± 10% range due to the rest of the *end-to-end* query costs (the eventual data access).

The conclusion from the study was that the "right" LSM-based spatial index to provide was simply the R-tree, as R-trees work for both point and non-point data. We added a small improvement for their storage efficiency in the case of point data (not storing them as infinitely small bounding boxes in the index leaves ☺), and we made a change in how deletions were handled for LSM, making those changes in the master version of Apache AsterixDB and leaving all of the other index options out of the system as "those were for research." Any cases where they offered small performance gains would not be worth the software engineering pain to have them, and to maintain them, in the code base. This might be viewed as a "negative result," as we didn't have earth-shattering performance differences to report – but negative results are actually very important! This indicated that, at least in a full DBMS (or BDMS) query processing context, working further on new/different spatial index structures to improve performance would be, to borrow a Stonebraker phrase, "polishing a round ball."

C. To Make Sure It's Complete

It's one thing to come up with a new data structure or algorithm that works for your favorite use case. It's an entirely different thing to come up with one that can be used in practice, one that makes sense in practice and is worthwhile in practice! If your new index data structure can't be loaded, can't be updated, can't be made recoverable, or can't be made concurrent, it's not ready for prime time in a database system. Finish it, and then let's talk.

My favorite example here is a bit of education that I got from Goetz Graefe, a long-ago Wisconsin PhD alumnus, an indexing and query processing expert, and later an ACM SIGMOD E.F. Codd award winner. As the handling editor for one of his "more than you ever wanted to know about X" survey papers, the one on B+ trees [26], I asked him to explain (both to me and to readers) why most real database systems stop after offering B+ trees. Linear hashing is one of my favorite access methods to teach; it seems like a terrific idea; why isn't it found in most database systems, since hashing is O(1) and B+ tree key lookups are O($\log_f N$), if there is an average of f keys per index page and N keys in total? His answer (which I am heavily paraphrasing) was enlightening (see Section 2.5 of [26] on "B-trees Versus Hash Indexes"): It is well-known how to efficiently load a B+ tree; it is ***not*** known how to do the same for Linear Hashing. Moreover, given a modest allocation of memory, their I/O costs in practice will be the same. Tell me again why you would want to add such a new, large component to your system, and solve its concurrency and recovery problems, and then maintain it? (Goetz 1, Mike 0.)

D. *To Make Sure It's Helpful*

You might have a great idea, one that today, if it were my idea, I would present and then ask: "What's not to love about that – how could you possibly not want that?" If you build your idea into an actual system, or build the system itself, and then get users, you will get feedback – often enlightening feedback. We have learned quite a bit in the AsterixDB effort by engaging with users who are doing things with our system and paying attention to what they need and what they find challenging.

As one example, UCI colleague Gloria Mark and one of her PhD students (Yiran Wang, now at Google) used AsterixDB as a data management and analysis aid for a study on stress and multitasking in college life [27]. We gained a great deal of insight from watching them use the system, answering their questions, and seeing what the stumbling blocks were. They needed to time-bin their data into various sized bins and to deal with the possibility that a given user activity might span bins (so they needed to allocate portions of such an activity to the relevant bins). We enhanced our temporal function support to deal with their requirements, as it was clear that other users could very well have similar needs. We also had support for CSV file import for data – they wanted export support, in addition, to round-trip their data in and out of the system in order to move it between analysis tools. We added that as well, based on their needs, as again it was a generally useful feature that we had simply missed thus far. And in addition, of course, we found a few bugs, query optimizer blind spots, and so on. Overall, they found the system useful – which was nice! – and we were able to improve it for the next users who might want to store and analyze multiple channels of temporal event data.

E. *To Get a Free Roadmap*

Last but not least, materializing your ideas in the form of an actual system that someone else will then use can lead to years of future entertainment. Once you have users, they have needs, and some of those needs are likely to point to new research directions or at to least major extensions or engineering challenges that are "future work worthy." As a young faculty member in Wisconsin, I sometimes felt like we were inventing and solving some of the problems that we worked on (particularly in our various simulation-based performance projects). "What next?" was a question that we had to answer periodically. Moreover, it was always a little disconcerting to see others following in our footsteps – there was a bit of a "no, wait – we made that up!" feeling that I would get on when reading/reviewing others' follow-up work. This was ***not*** my experience in industry – users of the systems I was involved in always had their own ideas and wish lists, so answering "what next?" meant choosing from a list of real needs and/or wishes instead of trying to guess what might be nice for users to have next.

VI. Initial Commercial Impact

Users and requirements are an excellent segway into the second-to-last part of this story, which is about the commercial adoption of AsterixDB technology. In November of 2018, Couchbase, Inc. released a new data-related service as part of the Couchbase Data Platform: Couchbase Analytics [28]. Under the hood, the Analytics service is based on the query processing and storage technology of Apache AsterixDB. Figure 7 summarizes AsterixDB's place in the Couchbase Data Platform world – in a nutshell, data and data changes in the Couchbase front-end data store are streamed in real time into the Couchbase Analytics backend, where it can then be sliced and diced in its natural (application schema) form using SQL++. The front-end Data Service is the data's home, and the data can be indexed and queried via the Index and Query services. The addition of Couchbase Analytics now allows users to conduct near real-time data analyses on an up-to-date copy of the data; this provides performance isolation, so heavy data analysis queries won't interfere with front-end operations and vice versa.

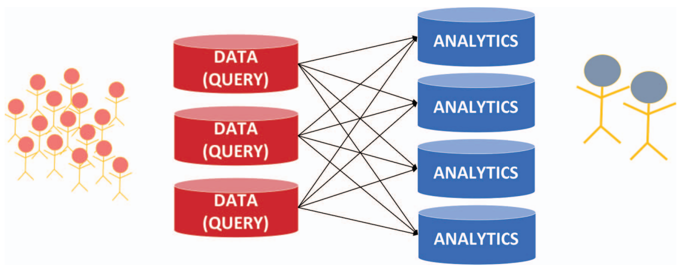

Figure 7. AsterixDB puts the ***A*** in NoSQL HTAP

Apache AsterixDB's commercial adventure began in about 2015, when a mutual friend introduced Rahim Yaseen, Couchbase's Senior VP of Engineering at the time, to this author. We quickly realized that we shared a declarative, general-purpose, query-based (i.e., database!) view of where NoSQL data platforms should be headed. Saeed moved on from Couchbase, but Ravi Mayarum, his replacement (and current Senior VP of Engineering and CTO), continued the discussions that had begun. The author was eventually given the opportunity to work with Couchbase in a consulting role to build a team inside Couchbase to create a new Analytics Service using AsterixDB as a starting point. This made sense because the existing Data, Index, and Query services were all aimed at use cases involving high volumes of small requests (operational throughput), while AsterixDB's parallel database-inspired architecture could provide the parallel query capabilities needed for more complex analytics over large volumes of JSON data [29].

The Couchbase discussions were also another motivator for the movement of AsterixDB into Apache. The AsterixDB team wanted to ensure that the code base would remain open and be accessible, without restrictions, to any/all comers. Once AsterixDB was in Apache, and licensed accordingly, Couchbase Analytics was then developed as an extension of Apache AsterixDB over a period of approximately two more years. Its core storage and query engines continue to be developed in open source, making the Couchbase developers some of the most prolific committers to Apache AsterixDB. The majority of the SQL++ work was done by Couchbase developers, in fact. Being an enterprise feature of the Couchbase Platform, for which customers pay money, Couchbase Analytics also has value-added features, primarily in the areas of cluster dynamics and failure handling and tooling, which are not a part of Apache AsterixDB. (Those features proved to be much more challenging than one might first imagine, particularly when starting with a system that was designed initially, for lack of manpower to do everything at once, for static cluster configurations and offline configuration changes.) The new Couchbase Analytics service is now starting to find its way into a variety of interesting industrial use cases, and the aforementioned "free roadmap" is already unfolding rapidly on both the commercial and research sides of the AsterixDB project.

VII. A Balancing Act

AsterixDB began as a research project, albeit one with full-system aspirations. Becoming an Apache "product," and now the basis of a commercial product, have led to an interesting ongoing balancing act. The Apache move meant that a code base that was once a playground for researchers could no longer be viewed that way – it needed to be cleaned up, with its research nooks and crannies being cleaned out. It also meant that a code base that had been co-managed by faculty (essentially a shared benevolent dictatorship) needed to transition to Apache-style community management. The Couchbase adoption raised the quality stakes much further; preparation for paying customers meant that the code base needed further hardening and a major makeover in terms of error handling and feedback. (Research projects tend to focus mostly on the "happy path.")

Today, Apache AsterixDB serves three user bases. The first is the open source community, who expect Apache-branded Big Data software to meet certain quality requirements, particularly as they mature over time. The second is Couchbase and its users, who have a very high quality bar and for whom "you get what you pay for" means something different than it does for open source users. The third are researchers and educators at universities and research institutes on multiple continents. It is extremely important to us that Apache AsterixDB can continue to serve that community as well, providing a basis for teaching "NoSQL done right" and for empirical, systems-oriented database research. We have seen other projects, when commercialized, lose that third role; we are committed to avoiding that outcome for AsterixDB if at all possible.

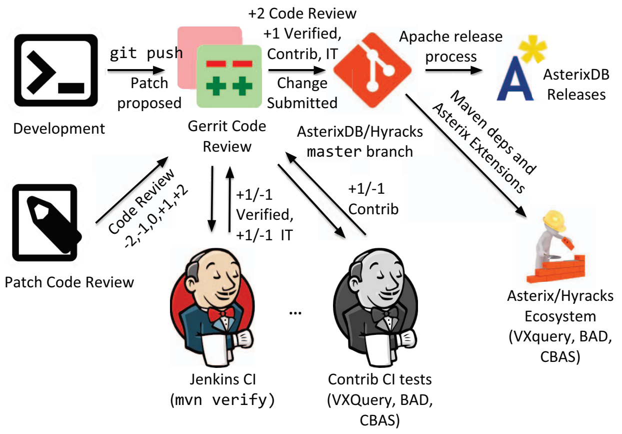

Figure 8. AsterixDB code management framework

Figure 8 shows how the Apache AsterixDB code base (which is a large Java code base) is currently being managed in order to meet the needs of all three kinds of users. Tooling-wise, the project uses git for source code management, Maven for compile and test purposes, IntelliJ and Eclipse (mostly) for IDEs, Jenkins for project automation and testing, Gerrit for code reviewing, and Docker for build isolation during tests. Changes to the AsterixDB/Hyracks master branch are carefully controlled. All changes have to be code reviewed by multiple committers, receiving +1's and at least one +2, before being applied, and there is a large (and growing) body of tests that are run on the various system components under various configurations;. The test automation system also votes on changes in order to avoid functional regressions. Changes with the potential to cause performance regressions are carefully performance-tested and discussed as part of the review process.

In addition to the core master code base, the code base supports a notion of recognized *extensions* that are based on AsterixDB and/or Hyracks code but extend it in their own ways. Three examples today are Apache VXQuery, BAD (Big Active Data), and CBAS (the Couchbase Analytics service). As mentioned in Section III, Apache VXQuery makes shared use of AsterixDB's Hyracks and Algebricks layers; it is a separate Apache project that offers a parallel XQuery engine for XML data. BAD is the UCI/UCR Big Active Data project, which extends AsterixDB's features with additional DDL and DML capabilities to support data-oriented pub/sub. CBAS is Couchbase Analytics, which we just discussed. All three of these extensions have test suites that run as part of the Apache AsterixDB continuous integration process in order to prevent extension-breaking changes to the master code base from going unnoticed. Informal extensions, like individual student MS and PhD research extensions, do not participate at this level – they are expected to work in their own git branches and sync with master frequently in order to avoid divergence. The results of such efforts are brought back into master, if at all, in small pieces after careful scrutiny and cleanup.

So far, the AsterixDB balancing act seems to be working. Having Couchbase committers has been tremendously beneficial to the project; the code base is much stronger today, and students are learning a great deal through their regular interactions with Couchbase's "seasoned professionals." Couchbase also benefits, as various open source university efforts enable the system to offer more features and improvements than it could if only Couchbase manpower were available; recent examples include storage compression and much-improved parallel sorting. The research community benefits as well; there are AsterixDB-related projects in locations around the globe, including China, Korea, India, Saudi Arabia, Germany, and Norway, to name a few, in addition to its use at various (currently mostly West coast) universities in the US. It has also been beneficial for teaching; its use at the University of Washington has been especially useful, to them as a NoSQL system with a full query language, and to AsterixDB because the Washington faculty and students have been a terrific source of user (usability) feedback. Universities continue to benefit from AsterixDB's availability for use as a software laboratory; numerous MS and PhD thesis have come out of the project [30] and continue to use branches for research purposes.

VIII. Closing Thoughts

Today, AsterixDB is "mid-flight" – it is kind of in its teenage years as a software project. Note that database systems can take a long time to mature; the UC Berkeley Postgres project, from which PostgreSQL came, was starting around the time of my PhD graduation (!) So far it has been an interesting, fruitful, and fufilling ride. Building systems in academia is challenging, as this paper has described, but creating working systems is also an exciting and satisfying undertaking. There are a number of models that others have used, including the "Patterson Berkeley lab model" (start a big project with industrial support, give it an explicit 5-year lifetime, and then move on) and the "Widom Stanford project model" (start a smaller project by adding one key new thing to existing systems, e.g., relaxing schemas or adding streams). Here I have shared the UCI/UCR AsterixDB model; time will tell how this model will fare in the end. Meanwhile, my closing advice to young faculty would be: Do build systems, do have fun in the process, and do set a high ("BMW") standard for publishing – the system and the lessons are the point, and the resulting papers are for sharing your learnings.

Acknowledgment

The author would like to thank all of the faculty, staff, students, and software professionals who together created Apache AsterixDB as it exists today. It has very much been a team effort (and adventure) since the very beginning. The AsterixDB effort was supported at UCI and UCR by an initial UC Discovery grant, by NSF IIS awards 0910989, 0910859, 0910820, 0844574, and by NSF CNS awards 1305430 and 1059436. It has also received industrial support from Amazon, eBay, Facebook, Google, HTC, Infosys, Microsoft, Oracle Labs, and Yahoo! Research and it benefits from ongoing software contributions from Couchbase.

References

[1] W. A. Wulf and G. Bell, "C.mmp: a multi-mini-processor," in *AFIPS Fall Joint Computing Conference (2)*, 1972, pp. 765–777. [Online]. Available: https://doi.org/10.1145/1480083.1480098

[2] R. J. Swan, A. von Bechtolsheim, K. Lai, and J. K. Ousterhout, "The implementation of the Cm* multi-microprocessor," in *American Federation of Information Processing Societies: 1977 National Computer Conference, June 13-16, 1977, Dallas, Texas, USA*, ser. AFIPS Conference Proceedings, vol. 46. AFIPS Press, 1977, pp. 645–655. [Online]. Available: https://doi.org/10.1145/1499402.1499516

[3] A. K. Jones, R. J. C. Jr., I. Durham, K. Schwan, and S. R. Vegdahl, "StarOS, a multiprocessor operating system for the support of task forces," in *Proceedings of the Seventh Symposium on Operating System Principles, SOSP 1979, Asilomar Conference Grounds, Pacific Grove, California, USA, 10-12, December 1979*, M. D. Schroeder and A. K. Jones, Eds. ACM, 1979, pp. 117–127. [Online]. Available: https://doi.org/10.1145/800215.806579

[4] J. K. Ousterhout, D. A. Scelza, and P. S. Sindhu, "Medusa: An experiment in distributed operating system structure," *Commun. ACM*, vol. 23, no. 2, pp. 92–105, 1980. [Online]. Available: https://doi.org/10.1145/358818.358823

[5] M. L. Brodie, Ed., *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*. ACM/Morgan & Claypool, 2019. [Online]. Available: https://doi.org/10.1145/3226595

[6] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna, "GAMMA – A high performance dataflow database machine," in *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings.*, W. W. Chu, G. Gardarin, S. Ohsuga, and Y. Kambayashi, Eds. Morgan Kaufmann, 1986, pp. 228–237. [Online]. Available: http://www.vldb.org/conf/1986/P228.PDF

[7] M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, J. E. Richardson, E. J. Shekita, and M. Muralikrishna, "The architecture of the EXODUS extensible DBMS," in *On Object-Oriented Database Systems*, 1991, pp. 231–256.

[8] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling, "Shoring up persistent applications," in *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '94. New York, NY, USA: ACM, 1994, pp. 383–394. [Online]. Available: http://doi.acm.org/10.1145/191839.191915

[9] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirzadeh, V. J. Tsotras, R. Vernica, J. Wen, and T. Westmann, "AsterixDB: A scalable, open source BDMS," *PVLDB*, vol. 7, no. 14, pp. 1905–1916, 2014. [Online]. Available: http://www.vldb.org/pvldb/vol7/p1905-alsubaiee.pdf

[10] T. Kim, A. Behm, M. Blow, V. Borkar, Y. Bu, M. J. Carey, M. Hubail, S. Jahangiri, J. Jia, C. Li, C. Luo, I. Maxon, and P. Pirzadeh, "Robust and efficient memory management in Apache AsterixDB," 2019, submitted for publication.

[11] S. Alsubaiee, A. Behm, V. R. Borkar, Z. Heilbron, Y. Kim, M. J. Carey, M. Dreseler, and C. Li, "Storage management in AsterixDB," *PVLDB*, vol. 7, no. 10, pp. 841–852, Jun. 2014. [Online]. Available: http://dx.doi.org/10.14778/2732951.2732958

[12] V. R. Borkar, M. J. Carey, R. Grover, N. Onose, and R. Vernica, "Hyracks: A flexible and extensible foundation for data-intensive computing," in *ICDE*, 2011, pp. 1151–1162.

[13] Y. Bu, V. R. Borkar, M. J. Carey, J. Rosen, N. Polyzotis, T. Condie, M. Weimer, and R. Ramakrishnan, "Scaling Datalog for machine learning on Big Data," *CoRR*, vol. abs/1203.0160, 2012.

[14] V. Borkar, Y. Bu, E. P. Carman, Jr., N. Onose, T. Westmann, P. Pirzadeh, M. J. Carey, and V. J. Tsotras, "Algebricks: A data model-agnostic compiler backend for Big Data languages," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: ACM, 2015, pp. 422–433. [Online]. Available: http://doi.acm.org/10.1145/2806777.2806941

[15] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using MapReduce," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, 2010, pp. 495–506. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807222

[16] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. J. Tsotras, "ASTERIX: towards a scalable, semistructured data platform for evolving-world models," *Distributed and Parallel Databases*, vol. 29, no. 3, pp. 185–216, 2011.

[17] M. J. Carey, S. Jacobs, and V. J. Tsotras, "Breaking BAD: a data serving vision for Big Active Data," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016*, 2016, pp. 181–186. [Online]. Available: http://doi.acm.org/10.1145/2933267.2933313

[18] K. W. Ong, Y. Papakonstantinou, and R. Vernoux, "The SQL++ semi-structured data model and query language: A capabilities survey of SQL-on-Hadoop, NoSQL and NewSQL databases," *CoRR*, vol. abs/1405.3631, 2014.

[19] D. Chamberlin, *SQL++ for SQL Users: A Tutorial*, September 2018. (Available via Amazon.com.).

[20] V. Borkar, M. J. Carey, and C. Li, "Inside "Big Data management": Ogres, onions, or parfaits?" in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12, New York, NY, USA, 2012, pp. 3–14.

[21] M. Stonebraker and U. Cetintemel, "One size fits all: An idea whose time has come and gone," in *Proceedings of the 21st International Conference on Data Engineering*, ser. ICDE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 2–11. [Online]. Available: https://doi.org/10.1109/ICDE.2005.1

[22] CCC Blog on Jeff Naughton's ICDE Keynote, http://www.cccblog.org/2011/05/16/jeff-naughtons-icde-2010-keynote/.

[23] Y. Kim, T. Kim, M. J. Carey, and C. Li, "A comparative study of log-structured merge-tree-based spatial indexes for Big Data," in *ICDE*. IEEE Computer Society, 2017, pp. 147–150.

[24] B. Liskov, A. Snyder, R. R. Atkinson, and C. Schaffert, "Abstraction mechanisms in CLU," *Commun. ACM*, vol. 20, no. 8, pp. 564–576, 1977. [Online]. Available: https://doi.org/10.1145/359763.359789

[25] B. Liskov, "The Argus language and system," in *Distributed Systems: Methods and Tools for Specification, An Advanced Course, April 3-12, 1984 and April 16-25, 1985, Munich, Germany*, ser. Lecture Notes in Computer Science, M. W. Alford, J. Ansart, G. Hommel, L. Lamport, B. Liskov, G. P. Mullery, and F. B. Schneider, Eds., vol. 190. Springer, 1984, pp. 343–430. [Online]. Available: https://doi.org/10.1007/3-540-15216-4_17

[26] G. Graefe, "Modern B-Tree techniques," *Found. Trends databases*, vol. 3, no. 4, pp. 203–402, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1561/1900000028

[27] G. Mark, Y. Wang, and M. Niiya, "Stress and multitasking in everyday college life: An empirical study of online activity," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/2556288.2557361

[28] Couchbase Analytics web page, Couchbase, Inc., http:://www.couchbase.com/products/analytics.

[29] D. Borkar, R. Mayuram, G. Sangudi, and M. J. Carey, "Have your data and query it too: From key-value caching to Big Data management," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, F. Özcan, G. Koutrika, and S. Madden, Eds. ACM, 2016, pp. 239–251. [Online]. Available: https://doi.org/10.1145/2882903.2904443

[30] ASTERIX, http://asterix.ics.uci.edu.