Good afternoon. Today, I will be presenting my workshop paper entitled “Embrace the Challenges: Software Engineering in a Big Data World”. I have a lot of material to cover in a short amount of time, so let’s get right to it!

We live in an exciting time.

<advance>

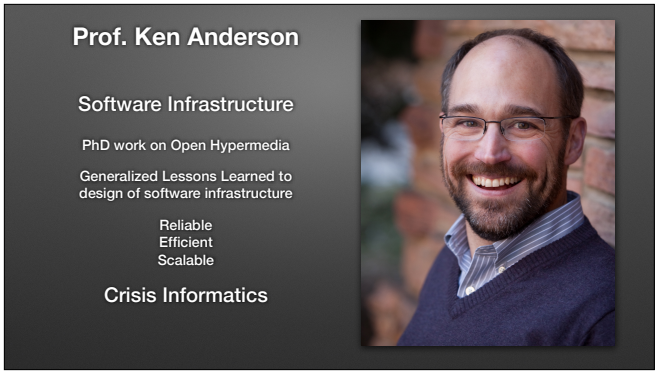
Today it is straightforward for a small group of software engineers to create software systems that generate, store, and process huge amounts of data. The tools and techniques behind this ability are collectively referred to as BIG DATA. I see that as a marketing term for work that has gone on for a long time in a variety of fields: software engineering, distributed systems, machine learning, etc.

<advance>

Despite this history, achieving this capability has made room for new research that has a long road ahead before we can truly say that we have a handle on the complexities involved in this domain. While a lot of work remains to be done, I personally find it to be an exciting area of study. My own area of expertise is software architecture and the design of software infrastructure.

<advance>

Thus my focus in this area of concern is on the design of what we call data-intensive software systems.



As you know, my name is Ken Anderson and I've been a professor of computer science for the last seventeen years with a long-time interest

<advance>

in the design of software infrastructure. Software infrastructure provides a set of services that can be used by a variety of tools to help users perform tasks. My PhD research at UC Irvine in the 90s

<advance>

looked at how to develop middleware that could provide hypermedia functionality to a wide range of applications. I learned a lot from designing that middleware and have since

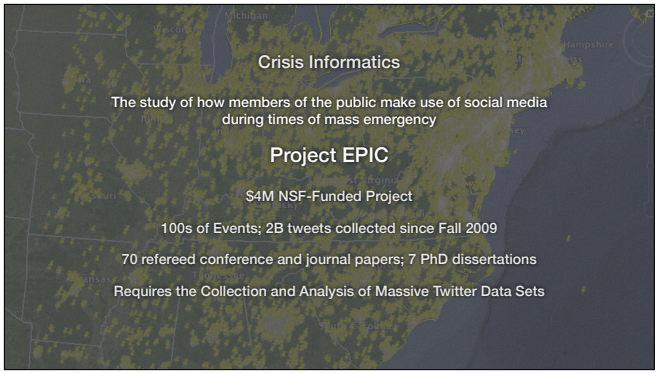
<advance>

generalized that work to provide services to a wide range of application domains; my research questions focused on what software architectural styles are best to deliver infrastructure that is

<advance>

reliable, efficient, and scalable.

The application domain where I test test these techniques and develop new software designs for data-intensive software systems is known as crisis informatics.



For instance, here is work that my group performed to collect and plot 70K tweets that were generated on the US Eastern Seaboard a few days before the landfall of Hurricane Sandy. We used these tweets to identify users of interest and we then collected all tweets that these users had ever tweeted leading to a dataset of nearly 300 million tweets. This data was then used to perform research on a wide variety of topics including the adoption of social media by fire and police departments and how they used their handles during a major crisis event and studying the evacuation behaviors of people directly in the storm's predicted path.

<advance>

As I just indicated, this area is broadly known as crisis informatics.

<advance>

It is the study of how members of the public make use of social media during times of mass emergency.

<advance>

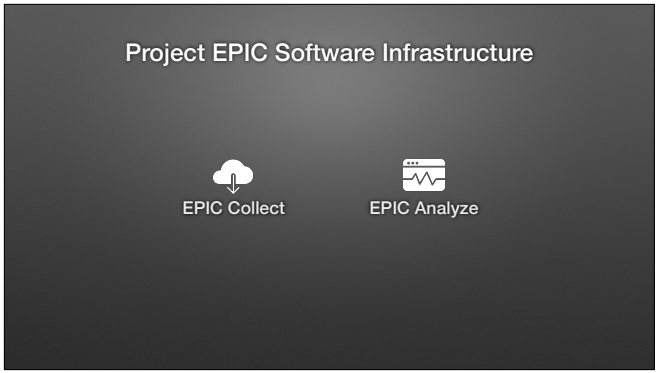
I work on a project know as Project EPIC which stands for empowering the public with information in crisis.

<advance>

It is a large, $4M NSF-project that

<advance>

has been collecting social media data on hundreds of crisis events since Fall



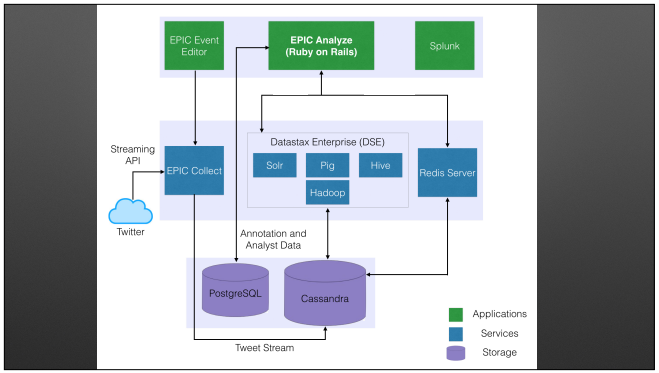
Project EPIC's software infrastructure rests on two main systems

<advance>

EPIC Collect and

<advance>

EPIC Analyze.



The software architecture of these systems is shown here. Our data collection system consists of a web application known as the EPIC Event Editor and the collection software itself known as EPIC Collect; EPIC Collect stores everything into a four node Cassandra cluster that is maintained in a professional data center.

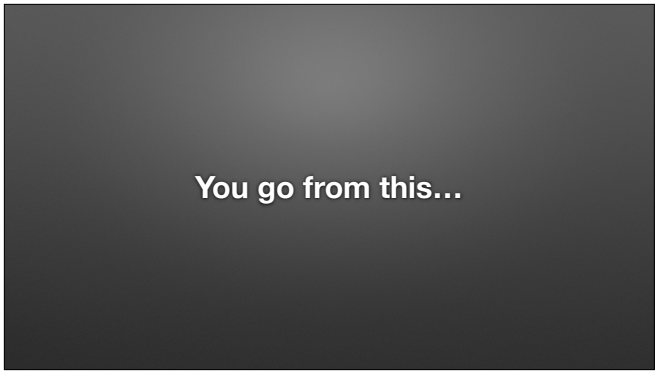
EPIC Analyze builds on that architecture by layering Solr, Pig, Hadoop, and Hive on top of Cassandra via the use of a software framework known as Datastax Enterprise. Each of these components can be used to help index, search, or process the large Twitter data sets stored in Cassandra. PostgreSQL is used to store annotations made by analysts while working with EPIC Analyze. EPIC Analyze is itself implemented as a Ruby on Rails web application that knows how to access all of the infrastructure provided by EPIC Collect and Datastax Enterprise. In addition, we make use of Redis to cache the results of frequently accessed queries and data.

Finally, we have the ability to integrate third-party data analysis tools, such as Splunk, but I don’t have time to discuss that particular aspect of our work today.

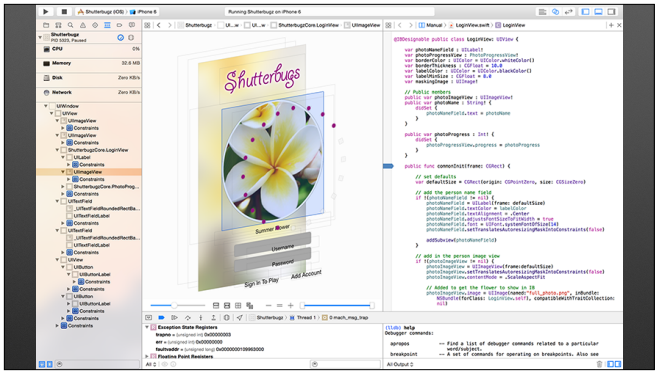
My workshop paper outlined six challenges that teams will encounter when working on the design, development, and deployment of data-intensive software systems. These challenges are:

The paper goes in depth on each of these issues. I will only have time today to treat them at a surface level.

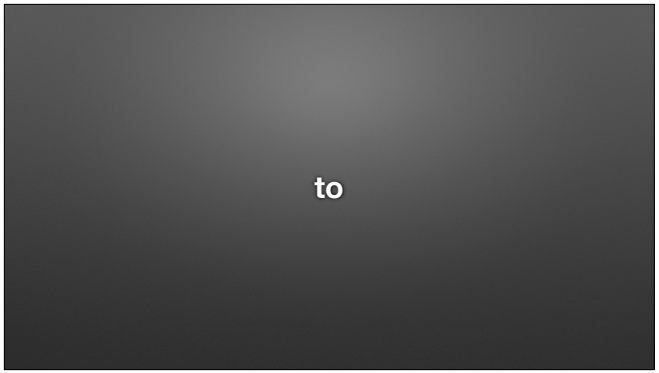
Lack of Developer Support

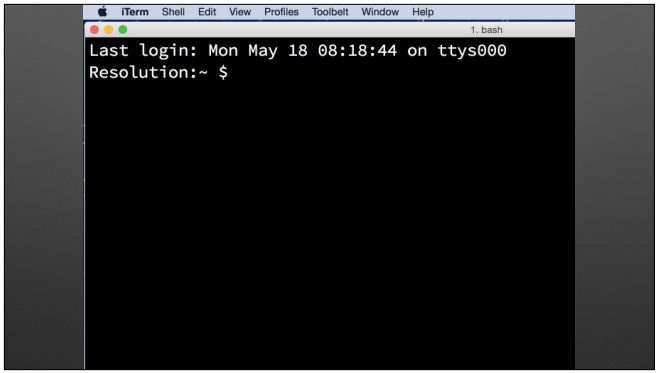


When moving from work on single applications to work on data-intensive software systems, you go from this



a full-fledged development environment with built in UI tools, static analysis tools, debuggers, editors, auto-complete, version control, advanced search and replace capabilities, automatic refactoring support, built-in testing support, integrated documentation, etc.

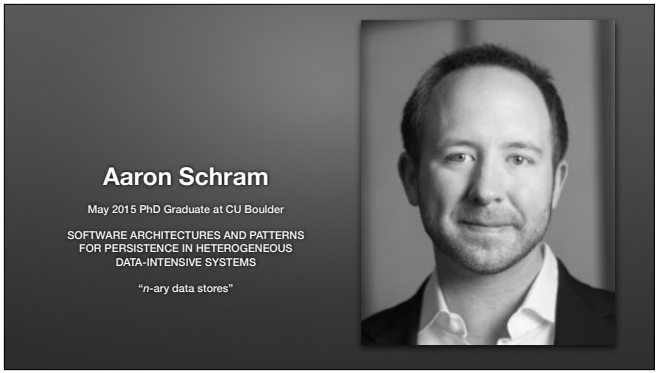

to



this. When working on data-intensive software systems, we're right back to a terminal screen with a blinking cursor: hadoop, Cassandra, mongo, etc. all have as their primary user interface a command line interface. The distributed systems community just assumes you know (and like!) to work this way. If you need tools on top of these frameworks to use them effectively, build them yourself!

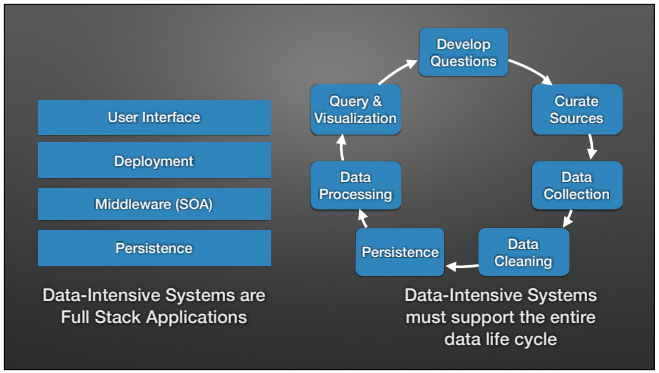
While this is an unfortunate and challenging situation, it represents a major research opportunity for software engineering. What modeling frameworks do developers of data-intensive systems need? What debugging tools would be most helpful? How do software engineers model and select the deployment options their system needs and can the actual deployment be automated or significantly reduced in complexity?

Over the next 10-20 years, we need to start developing the models and tools that will allow a team to work effectively in this space with as much power that we provide to developers working on single applications. As I mentioned earlier, it was encouraging to listen to Thomas's presentation as I would classify his work as addressing this very need.

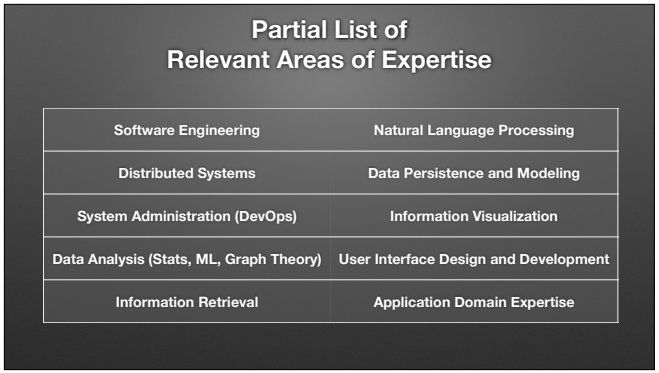


In our own work, my PhD student, Aaron Schram, just completed his PhD dissertation developing a software framework that makes it easy for systems to add or remove data stores. His work shows that in the same way that you can design your systems to take advantage of multiple cores, you can also design your system to take advantage of multiple data stores. We'll be working this summer to develop an ICSE paper based on his results and I hope to present his work in more detail in Austin next year.

A need for multidisciplinary teams



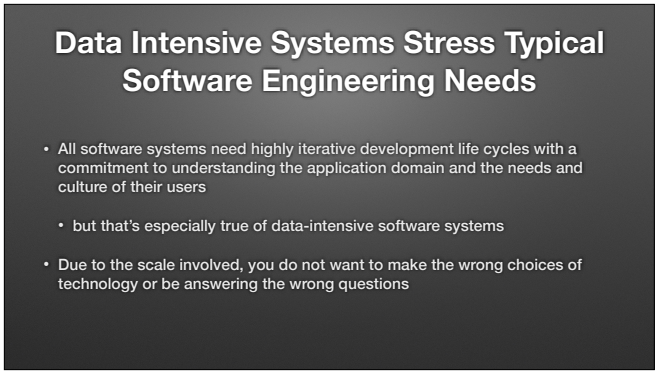
When building data-intensive software systems, you have to build systems that touch every aspect of the “stack”: from UI and deployment to middleware and persistence. Most data-intensive systems also have to support the full range of tasks related to the data life cycle from what questions are we going to answer to what data do we need to collect, to how are we going to store and process that data to the querying interface and visualizations that we provide our users; this info leads to new questions that kicks off the cycle once again.



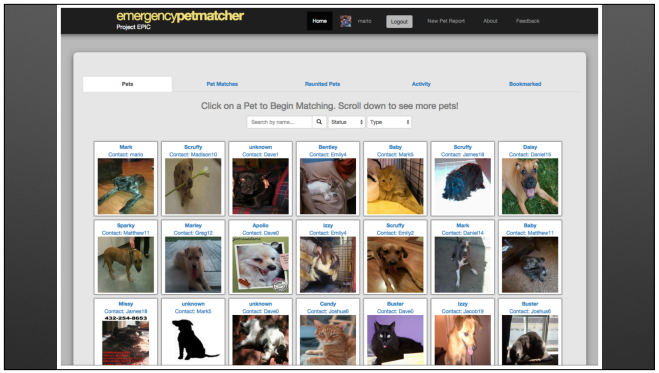
To address such a wide range of concerns, and pulling from my own experience with Project EPIC, we needed expertise in the following list of ten areas: <read them>

and I'm going to hedge my bets and call this a partial list and as a result, a challenge that engineers face in this domain is finding all the people needed to cover such a broad range of topics.

Iterative Life Cycles and a Commitment to the Domain



All software systems require iterative development life cycles and a commitment to understanding the application domain and its users but these common needs are stressed in data intensive software systems because if you get some aspect of the system wrong, it's really painful to recover. "What do you mean we have to reformat this 30 terabytes of data?" "The user told us they wanted to answer five main questions? Now they want to ask additional questions?"



In our work, we've been developing a system called Emergency Pet Matcher and we have recently published papers on the intensive user-centered design process that we made use of and our engagement with digital pet activists who want to help reunite lost pets with their owners after a disaster. We needed to make sure that we were supporting their actual workflows and not ones that we imagined. We needed to make sure that we were helping them answer the questions that arise each day. When we deploy this software, we have to be ready to respond to feedback and deploy changes right away or we risk disrupting the community and losing users.

Matching Frameworks with Requirements

Our approach to picking the right data store for EPIC collect was not based on a popularity contest. It was not a case of picking the most POPULAR data store… that's a horrible strategy for designing software systems.

Instead, we needed to understand our requirements and then find the persistence technology that met those needs exactly.

We needed flexibility so that when Twitter changes its metadata, our data store doesn't care.

We needed scalability since we never delete data sets. We have 2 terabytes of data so far and that isn't going away, its only going to grow.

We needed availability so that there is always at least one machine available to accept a tweet and persist it.

And, we needed reliability: once Twitter hands us a tweet, it is up to us to make sure that we don't lose it, since it might be impossible (without spending a lot of money) to find that tweet again.

So, we want to make sure we can store that tweet as soon as possible AND have it automatically replicated so if one of our servers goes down or a disk fails on us, we are assured not to lose any data.

Working in 2010 and 2011, the technology that met those needs was Cassandra. It is a columnar NoSQL data store that offers high availability, horizontal scalability, automatic replication, and a flexible data model. Cassandra is designed to run on clusters of machines; it can accept a read or write request on any server providing high availability; if you need more disk space, add another server; all data is replicated, so if a disk or node goes down, you still have access to all of your data while you wait for repairs, and Cassandra's data model is essentially a multi-level, distributed hash table that can store anything from movies (as Netflix does) to tweets (as Project EPIC does).

The important issue here is that the distributed systems community is not going to help you with this process. They are often focused on obscure aspects of the CAP theorem and a particular framework is their way of exploring all of the issues related to that particular portion of that design space. They will talk about the capabilities of a system but not necessarily with respect to how those capabilities impact software engineering concerns. You have to be clear with your own SE requirements so that you can then map those requirements onto the terms and capabilities used/discussed by the distributed systems crowd.

Easy Becomes Hard at Scale

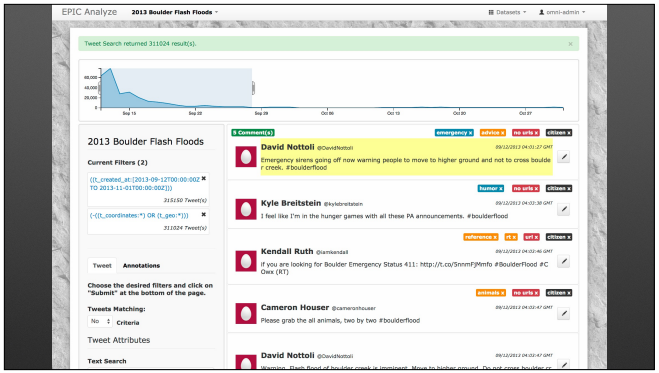
As a developer, moving into this space, you have to abandon pretty much everything you know and instead learn how to operate at scale. And everything easy becomes hard at scale…

…Even simple tasks like opening a file. I have a test I give to new students which is to point them at a 100 GB file and ask them to open it and tell me what's inside. This simple task will stump some people to the point of tears.

When you have lots of data, you can no longer just hand it around the various components of your system. If a query has ten million matches, there's no way you're passing 10M results across the line to display all of them in your user interface! You have to paginate, a concept foreign to many developers.

We like to keep our data models consistent and normalized but at scale we often have to abandon those concerns and settle for things like eventual consistency and data duplication.

In our own work, every tweet has an embedded user object: between the two entities there are over 70 different meta-data attributes associated with them and for the user, those attributes change with each tweet. Do you try to keep your "user table" up to date when tweets are streaming in at 100 tweets per second. The answer is no, you do not.

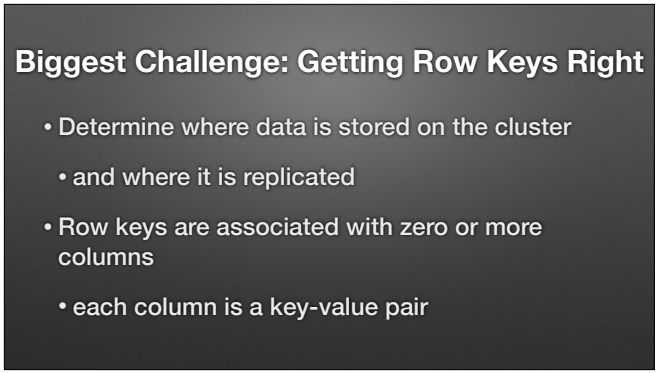


And, then there's sorting. Sorting… such a seemingly simply task. Take your data throw it at quick sort or bubble sort or merge sort, and call it a day right?

But imagine you've created a web app that allows users to browse millions of tweets. By default, these data sets are sorted by time. But an analyst looking at this result set, might say "Hmm, I wonder what other tweets David contributed to the data set?" and they want to sort the entire data set by user name. You can't pull millions of tweets into memory, sort them, and then redisplay with the browser focused on David's tweets. We tested that approach of sorting tweets in memory on a data set that consisted of just 200K tweets and it took seven minutes to bring all the tweets across the network and into main memory on the client machine, sort them, and display the result. If they wanted to then sort by retweet count, that would take another seven minutes to sort on that dimension.

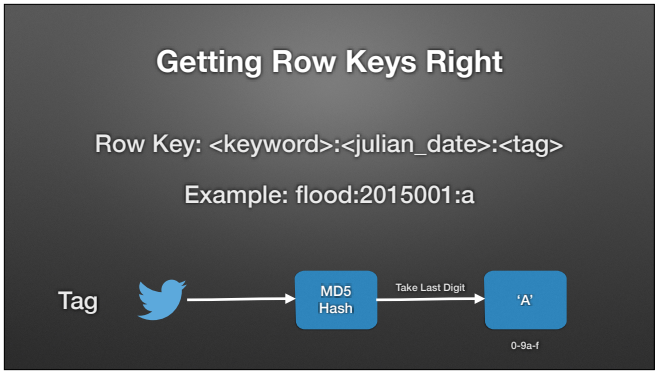
So, sorting becomes really hard. You have to do it ahead of time and create all sorts of indexes to make it possible for you to respond to these sort requests at interactive speeds.

Data Modeling. I like to say that data modeling is wicked hard and it is especially unforgiving when operating at scale. Not only do you have to get a structure that matches the types of queries that you want to answer but you have to make modeling choices that won't adversely impact the performance of the frameworks that you are using. Take EPIC Collect and our choice of Cassandra for instance.



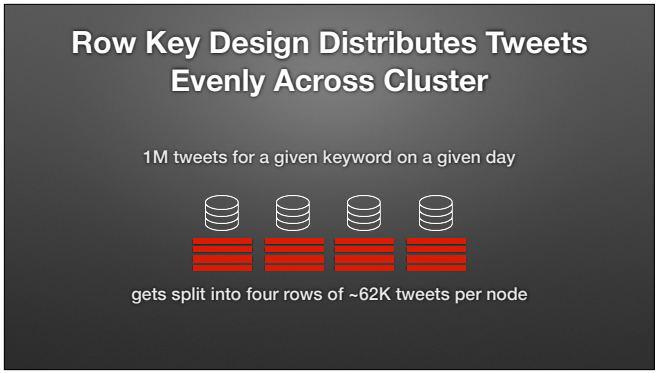
With Cassandra, the biggest challenge is designing the “row key” that is used to store data in column families. Row keys determine where data is stored on the cluster and what gets replicated on the cluster. Each row key can be associated with any number of columns; each column is a key-value pair.

You need your row keys to be grounded in the application domain for easy lookup and retrieval and you want to design them such that rows do not get too “wide” or too “narrow”, corresponding, in our case, to too many tweets on a particular row or too few.

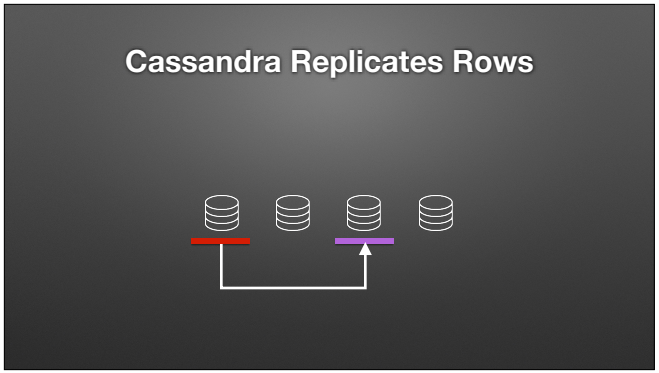


For our tweets, we use a compound row key that stores data in the following way: keyword, julian date, tag. For any individual row key this means that tweets stored in this row contained this keyword and were collected on this day.

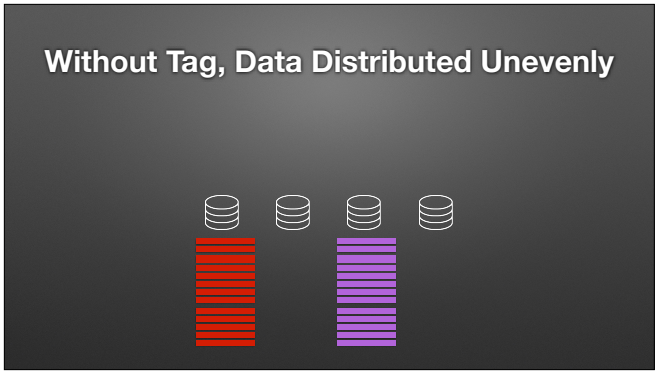
The tag represents a bucket; for each keyword/day combination we have 16 buckets. The tag is determined by computing a MD5 hash of the entire JSON object that Twitter gives us for a tweet and taking the last digit of that hash. We then store that tweet in that particular bucket.



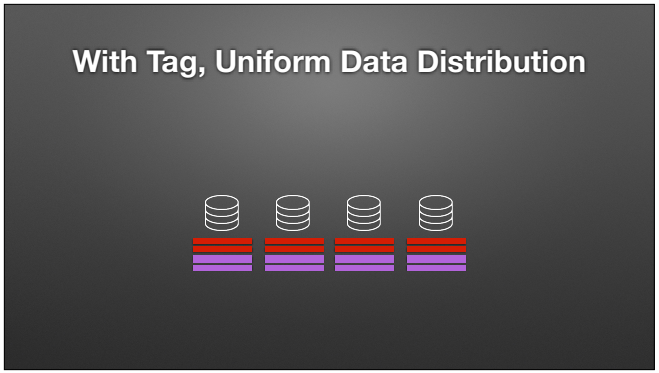
That means if we collected 1M tweets for a given keyword on a given day, we will have produced 16 different row keys in Cassandra each containing roughly 62 thousands tweets and Cassandra would ensure that these rows are spread evenly across the cluster. And, this is much better than trying to store a single row of 1M tweets on just one node of the cluster!



This is good because when we ask Cassandra to replicate our tweets, it performs replication at the level of row. So, when it replicates a row of tweets from one node to another, we get better performance if that row consists only of 1000s of tweets rather than millions.



If we didn't make use of this tag-based approach, we would see decreased performance because all the tweets for a given keyword/day combination might end up on a single node and then replicated to just one other node. You could have many nodes in your cluster sit idle while two nodes melt down under the load.



With the tag, we evenly distribute the load across the entire cluster both for primary rows and their replicas as shown here and keep our cluster effectively utilized.

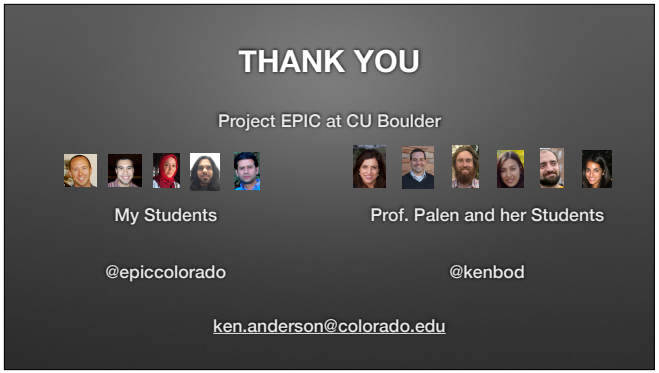
nepal:2015115:0	47125 tweets
nepal:2015115:1	47073 tweets
nepal:2015115:2	47016 tweets
nepal:2015115:3	46991 tweets
nepal:2015115:4	47000 tweets
nepal:2015115:5	46896 tweets
nepal:2015115:6	47215 tweets
nepal:2015115:7	47028 tweets
nepal:2015115:8	46795 tweets
nepal:2015115:9	46984 tweets
nepal:2015115:a	46744 tweets
nepal:2015115:b	46996 tweets
nepal:2015115:c	46833 tweets
nepal:2015115:d	46976 tweets
nepal:2015115:e	46769 tweets
nepal:2015115:f	47220 tweets

~750K tweets spread evenly across all 16 rows

Here's an example from our data collection of the 2015 Nepal Earthquake, one of our keywords was nepal and as you can see, we collected 750K tweets for that keyword during the first day of the event.

But, as you can also see, those tweets were fairly evenly split between all sixteen buckets and thus were evenly spread out across our cluster. When these rows were replicated, it was easier to perform that replication on rows of ~47K tweets each rather than one big row of 750K tweets

This has been a whirlwind tour through the six challenges that I have identified as critical concerns when working on the design and development of data-intensive software systems. In the paper, I go into more detail about each challenge and try to present examples taken from our work on Project EPIC both in terms of the problems encountered as well as some of the techniques we used to address those problems. It is my hope that other designers and developers working in this space will find these insights useful with respect to their own work.



Before I close, I would just like to acknowledge the work of my students at CU Boulder along with the other students on Project EPIC who work with Prof. Leysia Palen and drive all of the software engineering research that we do based on the needs of their own research that is focused on data analysis within the crisis informatics domain.

And, with that, I thank you for your attention and would be happy to answer any questions.