The following content is provided under a Creative Commons license. Your support will help MIT OpenCourseWare continue to offer high quality educational resources for free. To make a donation, or to view additional materials from hundreds of MIT courses, visit MIT OpenCourseWare at ocw.mit.edu.

JEREMY KEPNER: All right, we're going to get started here, everyone. And I know we have a lot going on here at the lab today, and I'm very pleased that those of you decided to choose us today over all the other things that are going on. So that's great, I'm very pleased. I wanted to say I think I've provided feedback to everyone who submitted their earlier first homework over the weekend, and I was very impressed and pleased by those of you who did the homework. I think you really embraced the spirit of the homework very well.

I know the homework that just happened is one that kind of really builds on that. For those of you who didn't do the first one, then you're kind of left out a colon. But I would like to give my congratulations for those who took on that second homework, which was significantly-- definitely required a fair amount of contemplation to sort of attempt at homework. So I want to thank those of you who did that.

So we're going to get started here. And so we're going to bring up the slides here. So we're going to be getting a little bit more. We're kind of entering the second of the three sections. We had these first three courses here that really was the first part.

Did a lot with motivation and a lot with theory, and now we're going to do stuff that's a little bit more-- you know, we'll be continuing that trend. We'll still always have plenty of motivation and a little bit of theory, but getting more and more towards things that feel like the things you might actually really do. And so we're kind of entering into that next phase of the course here. So let's get started here. So this is the lecture 03.

Pull up an example here from some real data, and talking about that. Here is the outline of the course. Actually just before we're doing that, sort of somewhat mandatory for those of you who are just joining us or are watching on the web and have jumped to add to this lecture, the title of the class is Signal Processing on Databases, is two words that we don't normally see together, or two phrases.

Signal processing, which is really alluding to Detection Theory, Formal Detection theory, and

its linear algebraic foundations. And databases, which is really alluding to dealing with strings, and unstructured data, and graphs. And so we're, in this course, talking about a technology, D4M, which allows us to pull these two views together. And so again we're now on the lecture 03, and we will continue on this journey that we have started.

Here's the outline. A little bit of history, I'm going to talk about how the web became the thing it is today, which I think talks a lot about why we invented the D4M technology. So it's going to talk about the specific gap that it fills, and talk a little bit about the D4M technology, some of the results that we've had. And then, hopefully today, the demo is more substantive, and so we'll spend more time on that than we did last time, where last time was more serious than the demo was really fairly small.

So for those of you who don't remember back in the ancient primordial days of the web, back in the early 90s, this is what it looked like. So if I talk about the hardware side of the web, it was quite simple. So there we go. The hardware side of the web was all Sun boxes.

We all had our Sun workstations, and there were clients, and there were servers, and there were databases. And the databases that we had at the time were SQL databases. Oracle was sort of really kind of just beginning to get going then. Sybase was sort of a dominant player at that stage.

This is sort of an example of the very first modern web page, or website, that was built. It was built by-- I'm sure there are other people who are doing very similar things at same time, so it's among the first. It's difficult to-- not going to be like Al Gore and say I invented the internet. But a friend of mine, Don Beaudry and I, were working for a company called Vision Intelligence Corporation, which is now part of GE.

And we had a data set and a SQL database, and we wanted to make it available. And there are a lot of thick client tools that came with our Sybase package for doing that, but we found them a little bit clunky. And so a friend of mine, Don, he downloaded this-- he said, hey I found this new software. I found a beta release of this software called NCSA Mosaic.

And so for those who don't know, NCSA Mosaic was the first browser, and it was invented at the National Center for Supercomputing Applications. And they released it. And it was awesome. It changed everything.

Just so you know, prior to that, and not to disrespect anybody who invented anything, but

HTTP and HTML were dying. They had actually come out in 1989, and everyone like, this is silly. What's this? Why do I have to type this HTTP and then this HTML, and it's just very clunky and stuff like.

That and it was getting its clock pretty well cleaned by another technology called Gopher, which was developed in the University of Minnesota, which was a much easier interface. The server was a lot better. It was a lot easier to do links then in HTML. And it was just a menu driven, text based way to do that.

Well Mosaic created a GUI front end to HTML and allowed you to have pictures, and that changed everything. That just absolutely-- and it just sort of caught on like wildfire. But it was just a beta at that point in time, and I don't think you could really do pictures, but it rendered fonts nicely.

And Gopher did not. Gopher was like just plain text. So you could have like underline, in bold, and the links were very clear. You could click at them. You could even have indents and bullets. You could imagine organizing with something that would look like a web page today.

And so we had that, and they're like, oh well we'll talk to our database using that. We'll use you HTT put to post a request. The Mosaic server was so bad, we actually just used the Gopher server to talk to the Mosaic browser.

And so we would do puts, and then we used a language called perl, which many of you know, because it actually had a direct binding to Sybase. And so we would take that, format it, and create SQL queries. And then it would send us back strings, which we would then format as HTML, and input to the browser, and it was great. It was like very nice, and it was sort of the first modern web page that we did in 1992. And essentially a good fraction of the internet kind of looks like this today.

Our conclusion at that time was that this browser was a really lousy GUI, and that HTTP was really trying to be a file system, but it was a really bad file system. And Perl was really bad for analysis. And SQL really wasn't-- it's good for credit cards transactions, but it wasn't really good for data in and out.

So our conclusion was this is an awful lot of work just to use some documents, and it won't catch on. Our conclusion was it won't catch on. So with that conclusion, maybe I'm underscoring my judgment for the entire course that we decided this was a bad idea.

So after that we had sort of what we call the Cambrian Explosion of the internet, which is we had an explosion. On the hardware side, we had big, giant data centers merging servers and databases, which is great. We had all these new types of hardware technologies here, laptops, and tablets, and iPhones, and all that kind of stuff.

So the hardware technology went forward in leaps and bounds. And then the browsers really took off, all of them. We have all these different browsers. You can still see, I have the mosaic logo back there still a little bit, because they're still showing.

For those of you who don't know, the author of Mosaic and NCSA was a guy by the name of Marc Andreessen. He left NCSA and formed a company called Netscape, which actually, at a moment in time was the dominant-- they were the internet. They got crushed, although I think Marc Andreessen made a fair amount of money.

But they did open source all their software, and so they basically rewrote Mosaic from the ground up, and that became Mozilla and Firefox and all that. So that's still around today. Safari may have borrowed from this. I don't really know.

When Netscape was dominating the internet, Microsoft needed to-- first they thought the internet was going to be a fad. And then they're like, all right this is serious. We need a browser. And so Mosaic had actually been spun out of the University of Illinois into a company called Spyglass, which may actually still exist today. And then Microsoft bought the browser, which was the original NCSA Mosaic code, and that became in Internet Explorer. So it's still around today.

This is Chrome, and as far as I know chrome is a complete rewrite from the ground up. And probably some of the other, like Opera, and other browsers that around today are completely-- but they still use it. It's a browser to use.

HTML still uses the basic put get syntax. SQL, we're still using it today. We have all whole explosion of technologies, Oracle being very dominant, but my SQL, SQL Server, or other types of things, Sybase still around, very common.

And we still have two major servers out there, Apache and Windows. So Apache was the Mosaic server. And just so you know, Apache was-- it's a patchy server, meaning it was very buggy. So it was the a patchy server, and that became Apache.

So that was the new Mosaic server, became Apache because it was a patchy server. And I bet they had borrowed code from Gopher, so that's why the Gopher, I think, is still showing there a little bit. We've had a proliferation of languages that are very similar. So people, in terms of gluing these two together, Java, et cetera, all that type of stuff.

But the important thing to recognize is that this whole architecture was not put together, because somehow we felt the web browser was the optimal tool, or SQL was the optimal tool, or these languages are the optimal tool. They were just things that were laying down to could be repurposed for this job. And so that's why they're not necessarily optimal things. But nevertheless, our conclusion has to be slightly revised that it would not catch on, because obviously it did catch on. The desire to share information was so great that it overcame the high barrier to entry just to set up on a web page or something like that.

Now things have changed significantly. We're not the first people to observe that this original architecture is somewhat broken for this type of model. And so today the web looks a lot more like this. You see all of our awesome client hardware which is great.

We have servers and databases that have now moved in type of the data center into shipping containers and other types of things, which is great. Increasingly, our interfaces will look a lot more like video games. I would tell you that a typical app on the iPad feels a lot more like a video game than a browser, which is great, so we're really kind of moving beyond the browser is the dominant way for getting our information.

And companies like Google and others recognize that, if you're just presenting data to people, and you're not doing say credit card transactions, then a lot of the features of SQL in those types of databases are maybe more than you need, and you can actually make these sort of very scalable databases more simply that are called triple stores. And so Google Bigtable being the original sort of one. Riak is another. Cassandra is another. HBase is another. Accumulo, which is the one we use in this class, is another.

So this whole new technology space has really driven that and has made a lot more scalable. But unfortunately, in the middle here, we're still very much left with much of the same technology. And this is kind of the last piece, this sort of glue. And so our conclusion is now that it's a lot of work to view a lot of data, but it's a pretty great view, and we can really view a lot of data.

But we still have this middle problem. There's this gap. And so we're trying to address that

gap. And so we address that gap with D4M. So D4M is formally designed from the ground up to be something that is really designed to work on triples, the kind of data we store in these modern databases, and do the kind of analysis in less code than you would otherwise.

So D4M, which stands for Dynamic Distributed Dimension Data model, which is sometimes in quotes, databases for Matlab, because it happens to be written in Matlab. It connects the triple stores, which can view their data as very large, sparse matrices of triples with the mathematical concepts of associative arrays that we talked about last class. That's essentially what D4M does for you.

So that's kind of one sort of holistic view of the technology space that we're dealing with here. Another way to view at it, which I'm going to talk a little bit about how the technologies that we have brought this to you. So I'm going to tell you a little bit about LLGrid and what we do there, because you're using your LLGrid accounts for this, and so I want to tell you a little bit about that.

So LLGrid is the core supercomputer at Lincoln Laboratory. And we're here to help with all your computing needs. Likewise, we're here to help with your D4M needs. If you have a project, and you think it would be useful to use D4M in that project, please contact us.

You don't have to just take this class and think that you're on your own. I mean, we are a service. We will help you do the schemas, get the code going, get it sort of adapted to your problem. That's what we do.

LLGrid, we have actually well over this, but just so I don't have to keep changing the slide. We always say it's about 500 users, about 2000 processors. It's more than that. To our knowledge, it's really the world's only desktop interactive supercomputer.

It's dramatically easier to use than any other supercomputer. And as a result, as an organization, we have the highest fraction of staff using supercomputing on a regular basis of any organization, just because it's so easy. And it's also kind of been the foundation, the vision, for supercomputing in the entire state of Massachusetts. All of our ideas have been adopted by other universities in the state and moving forward.

Not really the technology that we're talking about today, but something that is really kind of the bread and butter of LLGrid, is our Parallel Matlab technology, which is now going to be celebrating its 10th anniversary this year. And it's really a way to allow people to do parallel

programming relatively easily. D4M interacts seamlessly with this technology, and so you will also be doing parallel database codes with this technology.

So we have this thing called Parallel Matlab. If you're an LLGrid user, you are entitled to, or you almost always get the book on Parallel Matlab. And it allows you to take things like matrices and chop them up very easily to do complicated calculations. And almost every single parallel program that you want to write can be written with just the addition of a handful of additional functions.

And we get to use what is called a distributed arrays parallel program model, which is generally recognized as by far the best of all the parallel programming models, if you are comfortable with a multi-dimensional arrays. Everyone at MIT is, so this is easy.

However this is not generally true. Thinking in matrices is not a universal skill, or universal training, and so it is a bit of-- sometimes we do say this is software for the 1%. So you know, it is not for everyone. But if you do have this knowledge and background, it allows you to do things very naturally. And as you know, we've talked about all of our associated arrays as looking at data in matrices, and so these two technologies come together very nicely.

And we are really the only place in the world where you can routinely use this model. This is the dominant model that people use when they do parallel programming, which is generally-- and you might say, well why do I say it's the best? Well we run contests. There's a contest every year where we bake off the best parallel programming models, and distributed arrays in high level environments wins every year. And so it's kind of the best model if you can understand them.

It wouldn't be a class on computing in 2012 unless we talked about the cloud, so we definitely need to talk about the cloud. And to make it simple for people, we're going to divide the cloud into sort of two halves. You can subdivide the cloud into many, many different components. There's what we call utility cloud computing, which is what most people use the cloud for.

So gmail, what are enterprise services, calendars online, all that kind of stuff, basic data sharing, photo sharing, and stuff like that, that's what we call utility cloud computing. You can do human resources on the cloud. You can do accounting on the cloud. All this type of stuff that's on the cloud, and it's a very, very successful business.

What we mostly focus on is what is called data intensive cloud computing. It's based on

Hadoop and other types of technologies. But even still, a lot of that is technologies that are used by large scale cloud companies, but they don't really share with you.

Google, although it's often called Hadoop, Google doesn't actually use Hadoop. Hadoop is based on a small part of the Google infrastructure. Google's infrastructure is vastly larger than that. And they don't just let anybody log into and say, oh go ahead mine all the Google data.

They expose it to services, allowing you to do some of that, but the core technology that they do, they don't expose. In all large companies, they don't expose that to you. They give you selected services and such. But this is really what we're focused on is, this data intensive computing.

If I want to further subdivided the cloud from an implementer's perspective, if you own a lot of computing hardware, I mean hundreds of kilowatts or megawatts of computing hardware, you're most likely doing one of these four things. You could be doing traditional supercomputing, which we all know and love, which is sort of closest to what we do in LLGrid. You could do a traditional database management system, so every single time you use a credit card, make a purchase, you're probably connecting to a traditional database managed system.

You could be doing enterprise computing, which mostly, this day and age, is run out of VMware. So this is the LLGrid logo. And so you could be doing that. So that's all the things I earlier mentioned. Or this new kid on the block, big data, which is all the buzz today, using java and MapReduce and other types of things.

The important thing to recognize is these four areas, each of these are multi tens of billions of dollars industry. The IT world has gotten large enough that we now see specialization down to the hardware level. The hardware, the processors, that these folks use is different. Likewise for these folks and these folks. You see specialization of chips, now, to these different types of things.

So each of these at the center of a multi-billion dollar ecosystem, and they each have pros and cons. And sometimes you can do your mission wholly in one, but sometimes you have to cross. And when you do cross, it can be quite a challenge, because these worlds are not necessarily compatible.

And I would say, just to help people to understand Hadoop, one must recognize that Java is

the first class citizen in the Hadoop world. The entire infrastructure is written in Java. It's designed so that if you only know Java, you can actually administer and manage a cluster, which is why it's become so popular. There are so many Java programmers, and now they just know Java, and they downloaded Hadoop, and they have rudimentary access to some computer systems they can sort of cobble together a cluster.

I should say the same thing occurred in the C in the Fortran commune in the mid 90s. There was a technology called NPI, in fact this is the NPI logo today, which allowed you, if you knew C, Fortran had rudimentary access on a network of workstations, you could create a cluster with NPI. And that really sort of began the whole sort of parallel cluster computing revolution. This does the same thing in Java, and so for Java programmers it's been a huge success in that regard.

Just so you know where we sit-in with that, so we have a LLGrid here, which has really made traditional supercomputing feel interactive and on demand and elastic, so that's one of the things that we do that's LLGrid, very unique. You launch your job. It runs immediately. That is different than the way every other supercomputing center in the country is run. Most supercomputing centers are run by, you write your program, and you launch it, and you wait until the queue says that it is run.

And in fact, some centers will even use, as a metric of success, how long their wait is. It's kind of like a restaurant, right? How many years do you have to wait to get a good reservation, or a good doctor right? And I remember someone saying, oh our system is so successful that our wait is a week.

You hit submit and you will have to wait a week to run your job. That's not really consistent with the way we do business around here. You're under very tight deadlines. So we created a very different type of system 10 years ago, which sort of feels more like what you people are trying to do in the cloud.

In the Hadoop community, there's been a lot of work to try and make databases. What these technologies give you is sort of efficient search, and so there are a number of databases that have been developed that sit on top of Hadoop, HBase being one, and Accumulo being another. Which to my knowledge is the highest performance open source triple store in the world, and probably will remain that way for a very long time. And so we will be using the Accumulo database.

And then we've created bindings to that. So we have our D4M technology, which allows you to bind you to databases. And we also have something called LLGrid MapReduce. MapReduce is sort of the core programming model within the Hadoop community. It's such a trivial model, it almost doesn't even deserve a name.

It's basically like, write a program given a list of files. Each program runs on each file independently. I mean it's a very, very simple model. It's been around since the beginning of computing, but the name has become popularized here, and we now have a way to do MapReduce right in LLGrid. It's very easy, and people enjoy it.

You can, of course, do the same thing with other technology in LLGrid, but for those people, particularly if you're writing job or-- not Matlab. We don't recommend people use this for Matlab, because we have better technologies for doing that. But if you're using Python, or Java, or other programs, then MapReduce is there for you.

As we move towards combining-- so our big vision here is to combine big compute and big data, and so we have a whole stack here as we deal with new applications, texts, cyber, bio, other types of things, the new APIs that people write, the new types of distributed databases. So it's really affecting everything. Just an example of the Hadoop architecture.

You've had this course. You should at least have a few minutes so that if someone asked, well what is Hadoop, you know what it does. This is the basic architecture. So you have a Hadoop MapReduce job. You submit it, and it goes to a JobTracker.

The JobTracker then breaks that up into subtasks, each of which has its own tracker. Those tasks then get sent to the DataNodes in the architecture, and it gets those DataNodes, and NameNode keeps track of the actual names of the things. Just a very simple a Hadoop cluster. This is the different types of the nodes that are in, and processes in architecture.

Hadoop's strengths and weaknesses, well it does allow you to distribute processing a large data. Best use case is, let's say you had enormous number of log files, and you decided you only wanted to search them once, for one string. That's the perfect use case for Hadoop. If you decide, though, that you would like to actually do more than one search, then it doesn't necessarily always make sense. So it's basically designed to run grep on an enormous number of files.

And I kid you not, there are companies today that actually do this, like at production scale. All

their analytics are done by just grepping enormous numbers of log files. Needless to say that the entire database community cringes, and rolls over in their graves if they are not alive, at this. Because we have solved this problem by indexing our data, and that allows you to do fast search. And people, that's why people have invented databases like HBase and Accumulo, which would sit on top of Hadoop because they recognize that, really, if you're going to search more than once, you should index your data.

Again, it is very scalable. It is fundamentally designed to have extremely unreliable hardware in it. So it is quite resilient. It does this resiliency at a cost. It relies on a significant amount of replication. Typically the standard replication in a Hadoop cluster is a factor of 3.

If you're in the high performance storage business, this also makes you cringe, because you're paying 3x for your storage. We have techniques like RAID, which allow us to do that. But again, the expertise required to set up a cluster and do 3x replication is relatively low, and so that makes it very appealing to many, many, many folks, and so it's very useful for that.

Some of the difficulties, the scheduler in Hadoop is very immature. Schedulers are very well defined. There's about two or three standard schedulers in the supercomputing community. They're each about 20 years old. They all have the same 200 features, and you need about 200 features to really do a proper scheduler.

Hadoop is about four or five years old, and it's got about that many features. And so you do often have to deal with collisions. It's easy for one user to hog and monopolise the entire cluster. You're often dealing with various overheads from the scheduler itself. And this is well known to the Hadoop community. They're working on it actively, and we'll see all that-- it's not an easy multi-user environment.

And as I've said before, it fundamentally relies on the fact that the JVM is on every node. Because when you're sending a program to every node, the interpreter for that must exist on every node. And by definition, the only language that you're guaranteed to have on every node in a Hadoop cluster is Java. Any other language, any other tool, has to be installed and become a part of the image for the entire system so that it can be run on every node. So that is something that one has to be aware of.

And we've certainly seen it's like, oh I wrote these great programs in another language. And it's like, well can't I just run them? It's like no, that doesn't exist. Even distributing a fat binary can be difficult, because it has to be distributed through the Hadoop Distributed File system.

All data is distributed there, and that is not-- you have to essentially write a wrapper program that then recognizes where to get your binary, and then exacts that thing and it can be tricky.

But the basic LLGrid MapReduce architecture simplifies this significantly, although this maybe doesn't look like it. Essentially you call LLGrid MapReduce. It launches a bunch of what are called mapper tasks. It runs them on your different input files. When they have done, they've created output files. And then runs another program. If you specified, it then combines the output.

So that's the basic model of MapReduce, which is you have what's called a map program and a reduced program, and the map program is given a list of files. It runs those programs on at a time. Each one of them generates an outpoint, and a reduce program pulls them all together.

All right, so that's a little tutorial on Hadoop, because I felt an obligation to explain that to you. We couldn't be a big data course without having doing that. But it's very simple, and very popular, and I expect it to maintain it's popular for a very long time.

I think it will evolve and get better, but for the vast majority of people on planet Earth, this is really the most accessible form of parallel computing technology that they have available. So we should all be aware of Hadoop, and its existence, and what it can do. Because many of our customers use Hadoop, and we need to work with them.

All right, so getting back now to D4M. I think I've mentioned a lot of this before. The core concept of D4M is multi-dimensional associative array. Again, D4M is designed to sort of overcomes. For those of us who have more mathematical expertise, we can do much more sophisticated things than you might be able to do in Hadoop.

Again, it allows you to look at your data in four ways at once. You can view it as 2D matrices, and reference rows and columns with strings, and have values that are strings. It's one to one with a triple store, so you can easily connect to databases. Again, it looks like matrices so you can do linear algebra. And also between the duality between adjacency matrices and graphs, you can also think about your data as graphs.

This is composable, as I've said before. Almost all of our operations that I perform an associated rate of return another associative array, and so we can do things like add them, subtract them, and them, or them, multiply them, and we can do very complicated queries very easily. And these can work on associative arrays. And if we're bound to tables, they can also

do that.

Speaking of tables, I've already talked about our schema in this class that we always use. So if your standard data might look like this, say this is a cyber record of a source IP, domain, and destination IP, this is sort of the standard tabular view. We explode it by taking the source, pending the value to it which creates this very large, sparse table here which will naturally go into our triple store.

Of course, by itself it doesn't really gain as anything, because most tables either are row based or column based. The databases that we are working with a row based, which allow you to do fast look up of row key. However, once we've exploded the schema, if we also store the transpose, we now can index everything here quickly and effectively to the user. It looks like we have indexed every single string in the database with one schema, which is very nice, very powerful stepping stone for getting results.

All right, I'm going to talk a little bit about what we have done here. So I'm just going to go over some basic analytics and show you a little bit more code. So for example, here is our table. So this could be viewed as the sparse matrix. We have various source IPs here. And I want to just compute some very elementary statistics on this data.

I should say, computing things like sums and averages may sound like not very sophisticated statistic. It's still extraordinarily powerful, and we are amazed at how valuable it is, because it usually shows you right away the bad data in your data. And you will always have bad data. And this is probably the first bit of value add we provide to almost any customer that we work with on D4M, is it's the first time anybody's actually sort of looked at their data in totality, and been able to do these types of things.

And so usually the first thing we do is like, all right we've loaded their data in schema, and we've done some basic sum. And then we'll say, do you know that you have the following stuck on switches? You know, 8% of your data all has this value in this column, which absolutely can't be right, you know? And 8% could be a low enough number that you might not just encounter it through routine traversal, but it will stand out in just the most rudimentary histogram, extremely.

And so that's an incredibly valuable piece of [INAUDIBLE]. Oh my goodness, they'll usually be like, there's something broken in our system. Or you'll be like, you have this column, but it's almost never filled in, and it sure looks like it should be filled in, so these switches are stuck off.

I encourage, when you do the first thing, to just kind of do the basic statistics on the data and see where things are working and where things are broken. And usually you can then either fix them. And most of the time when we tell a customer, it's like, we're just letting you know these are issues in your data. We can work around them. We can sort of ignore them and still proceed, or you can fix them yourselves.

And almost always they're like, no we want to fix that ourselves. That's something that's fundamentally broken in our system. We want to make sure that that data is correct. Because usually that data is flowing into all other places doing a being acted on, and so that's one thing we're going to do.

So we're going to do some basic statistics here. We're going to compute how many times each column type appears. We're going to compute how many times each type appears. We'll compute some covariance matrices, just some very, very simple types of things here.

All right, let's move on here. So this is the basic implementation. So we're going to give it a set of rows that we're looking at here. This could be very large. We have a table binding, T, so we say, please go and get me all those rows.

So we get the whole swath of rows, and that will return that as an associative array. And then, normally I shorten this is double logi, since the table always contain strings, it return strings values, the first thing we knew it was going to get rid of those, because we're going to do math. So we turn them into logicals, and then we turn them back into double so we can do math.

You can actually pass regular expressions. You can also do that starts with command. We want to get the source IP. We're just interested in source IP domain. And the first thing we do is find some popular columns. So we just type sum, and that shows up popular columns. If we want to find popular pairs, it's covariance matrix calculation there, or square in and find domains with many destination IPs.

An example in this data set is this was the most popular in the data set that we had. These were the most popular things that appeared here. And you can see, this is all fairly reasonable stuff, LLBean, lot of New England Patriots fans in this data set, a lot of ads, Staples. These are the types of things it shows you right away.

Here's the covariance matrix of this data. So as you can see, it's symmetric. Obviously there is a diagonal, and it has the bipartite structure. That is, there is no destination to destination IP links. And we have source IP the destination, domain, and all that type of stuff exactly.

The covariance matrix is often extremely helpful thing to look at, because it will be the first time it shows you the full interrelated structure of your data. And you can quickly identify dense rows, dense columns, blocks, chunks, some things where you want to look at, chunks that aren't going to be interesting because they're very, very dense or very, very sparse. Again a very, very just sort of survey type of tool.

A little shout out here to our colleagues in the other group, who developed structured knowledge space. Structured knowledge space is a very powerful analytic data set. I kind of tell people it's like the Google search where when you type a search key, it starts guessing what your next answer is.

But Google does that based on sort of a long sort of analysis of all the different types of searches people have done, and it can make a good guess. SKS does that much more deeply, in that it maintains a database on a collection of documents. And in this case, if you type Afghanistan, will then actually go and count the occurrences of different types of entities in those documents, and then show you what the possible next key word choices can be, so in the actual data itself, not based on a set of cached queries and other types of things.

I don't really know what Google actually does. I'm just guessing. I'm sure they do use some data for guiding their search, but this is a fairly sophisticated analytic. And one of the big sort of wins for we knew we were on the right track with D4M is that this has been implemented in hundreds or thousands of lines of job at SQL. And I'm going to show you how we implement it in one line in D4M.

So let's go over that algorithm. So here's an example of what my data might look like. I have a bunch of documents here. I have a bunch of entities, where an entity appears in a document. I have a dot. I'm going to have my associative array over here, which is matching from the strings to the reals.

And my two facets are just going to be two columns names here, so I'm going to pick a facet y1 and y2. So I'm going to pick these two columns. All right, so basically that's what this does. This says get me y1 and get me y2.

I'm using this bar here to say I'm going to knock away the facet name afterwards so I can and them together. So that gets me all the documents that basically contain both UN and Carl, and then I can go and compute the counts that you saw in the previous page there just by doing this matrix multiply. So and them, I transpose them, and I matrix multiply to get them together, and then there we go.

All right, so now I'm going to get into the demo, and I think we have plenty of time for that. I'll sort of set it up, and they'll take short break. People can have cookies, and then we will get into the demo, which really begins to show you a way to use D4M on real data.

So the data that we're going to work with here is the Reuters Corpus. So this was a corpus of data released in 2000 to help spur research in this community. We're very grateful for Reuters to do this. They gave the data to NIST who actually stewards the data sets. It's a set of Reuters news reports over this particular time frame, the mid 90s, and it's like 800,000 total.

And what we have done, we're not actually working with the straight Reuters data, we've run it through various parsers that have extracted just the entities, so the people, the places, and the organization. So it's sort of a summary. It's a very terse summarization of the data, if that. It's really sort of a derived product.

The data is power law, so if we look at the documents per entity, you see here that certain people, places, and organizations, appear. A few of them appear lots of places, and most of them appear in just a few documents, and you see the same thing when you look at the entities per documents. So there's essentially, could view this is computing the in degrees and the out degrees of this bipartite graphs. And it has the classic power law shape that we expect to see in our data.

So with that, we're going to go in the demo. And just to summarize, so just recall, the evolution of the web is really created a new class of technologies. We really see the web moving towards game style interfaces, triple store databases, and technologies like D4M for doing analysis, very new technology.

And just so that we, for the record, there is no assignment this week. So for those of you who are still doing the assignments, yay. The example, actually won't do both of these today, we'll just do one of them. Oh and I keep making this mistake. I'll fix it. I'll fix it later.

But it's in the examples directory. We've now moved on to the second subfolder, apps, entity

analysis. And you guys are encouraged to run those examples, which we will now do shortly. So we will take a short five minute break, and then continue on with the demo.