Lumberyard

Time Series Indexing At Scale

Today's speaker – Josh Patterson

- **josh@cloudera.com**
- Master's Thesis: self-organizing mesh networks
  - Published in IAAI-09: TinyTermite: A Secure Routing Algorithm
- Conceived, built, and led Hadoop integration for the openPDC project at TVA
  - Led small team which designed classification techniques for timeseries and Map Reduce
  - Open source work at http://openpdc.codeplex.com
- Now: Sr. Solutions Architect at Cloudera

Agenda

- What is Lumberyard?
- A Short History of How We Got Here
- iSAX and Time series Data
- Use Cases and Applications

cloudera

What is Lumberyard?

- **Lumberyard is time series iSAX indexing stored in HBase for persistent and scalable index storage**
- It's interesting for
  - Indexing large amounts of time series data
  - **Low latency** *fuzzy pattern matching queries* on time series data
- Lumberyard is open source and ASF 2.0 Licensed at Github:
  - https://github.com/jpatanooga/Lumberyard/

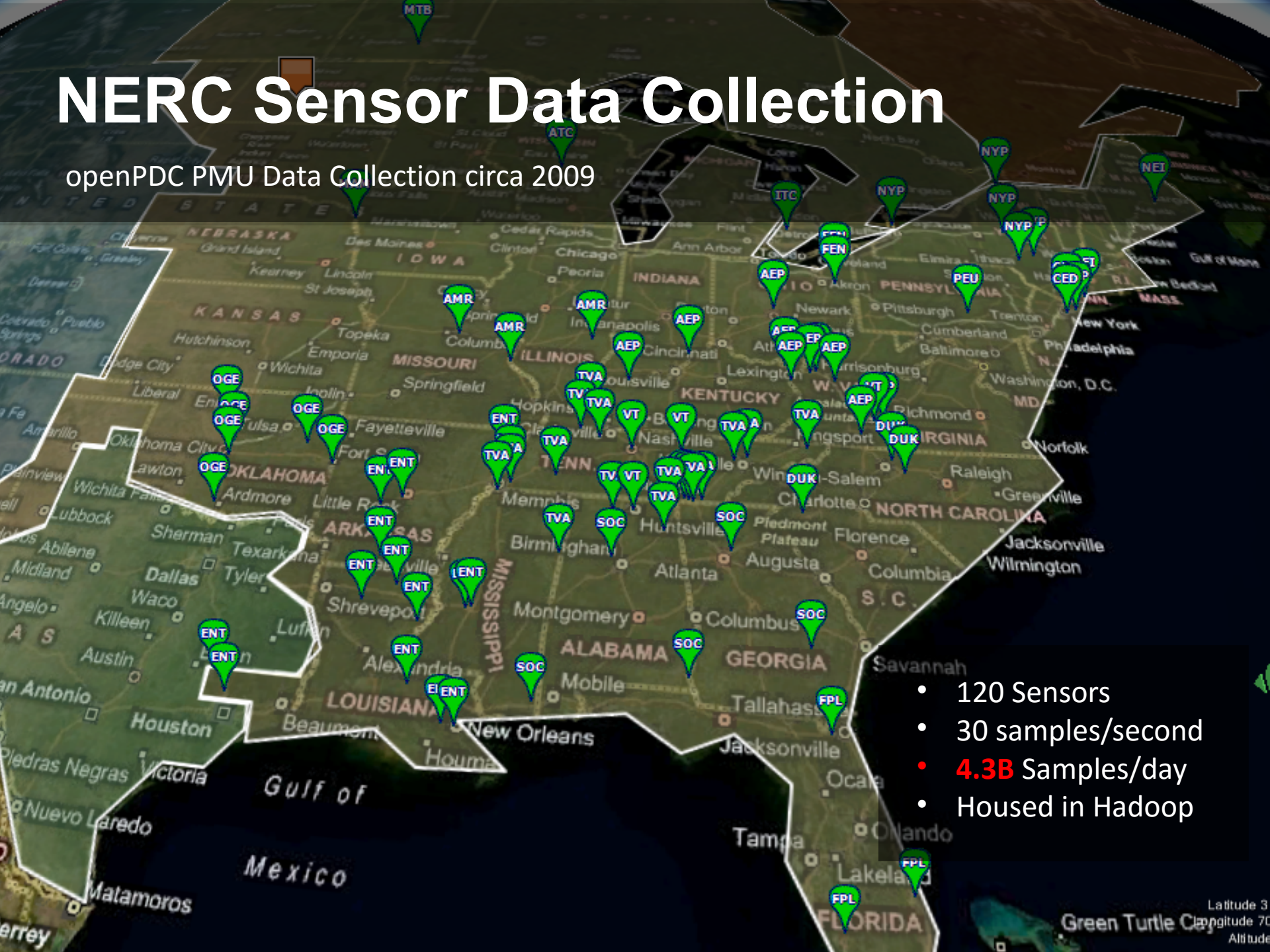
cloudera

A Short History of How We Got Here

cloudera

NERC Sensor Data Collection

openPDC PMU Data Collection circa 2009



- 120 Sensors
- 30 samples/second
- **4.3B** Samples/day
- Housed in Hadoop

Story Time: Keogh, SAX, and the openPDC

- NERC wanted high res smart grid data tracked
  - Started openPDC project @ TVA
    - http://openpdc.codeplex.com/
  - We used Hadoop to store and process time series data
    - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- Needed to find “unbounded oscillations”
  - Time series unwieldy to work with at scale
- We found “SAX” by Keogh and his folks for dealing with time series

cloudera

What is time series data?

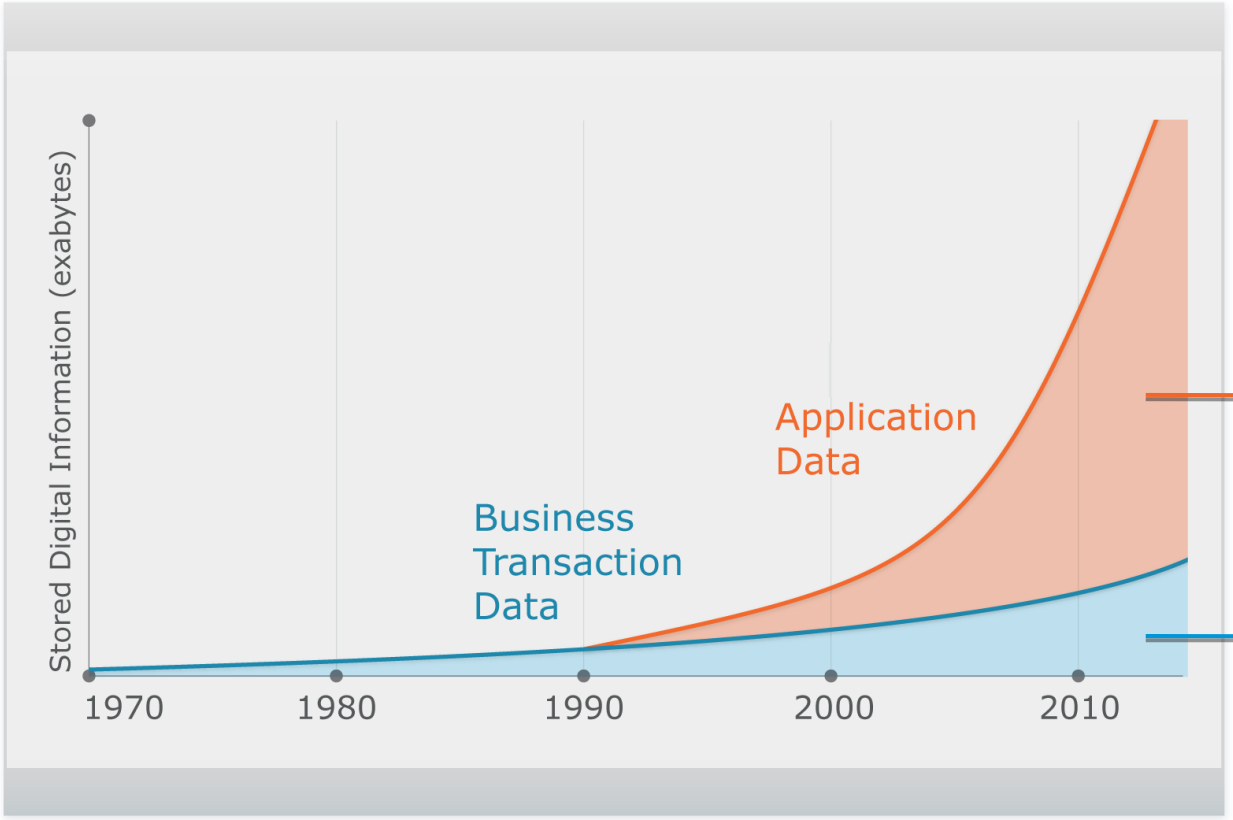
- Time series data is defined as a sequence of data points measured typically at successive times spaced at uniform time intervals
- Examples in finance
  - daily adjusted close price of a stock at the NYSE
- Example in Sensors / Signal Processing / Smart Grid
  - sensor readings on a power grid occurring 30 times a second.
- For more reference on time series data
  - http://www.cloudera.com/blog/2011/03/simple-moving-average-secondary-sort-and-mapreduce-part-1/

cloudera

Why Hadoop is Great for the OpenPDC

- Scenario
  - 1 million sensors, collecting sample / 5 min
  - 5 year retention policy
  - Storage needs of 15 TB
- Processing
  - Single Machine: 15TB takes 2.2 DAYS to scan
  - Hadoop @ 20 nodes: **Same task takes 11 Minutes**

Unstructured Data Explosion



(you)

Complex, Unstructured

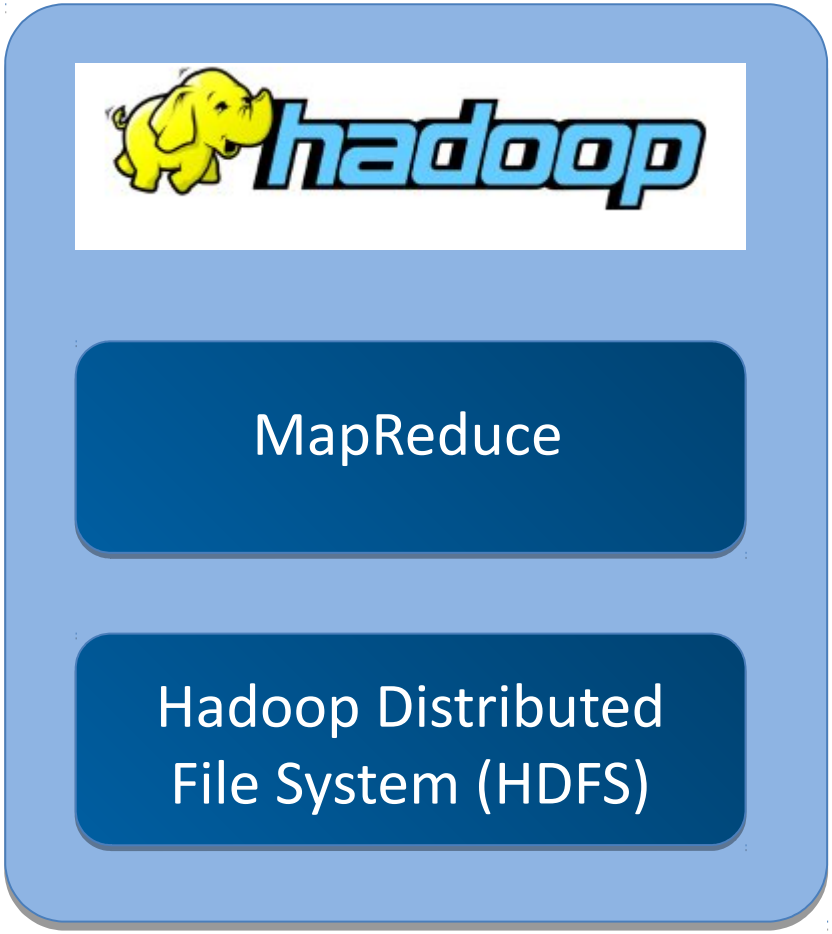
Relational

- 2,500 exabytes of new information in 2012 with Internet as primary driver
- Digital universe grew by 62% last year to 800K petabytes and will grow to 1.2 "zettabytes" this year

cloudera

Apache Hadoop

Open Source Distributed Storage and Processing Engine



- **Consolidates Mixed Data**
  - Move complex and relational data into a single repository
- **Stores Inexpensively**
  - Keep raw data always available
  - Use industry standard hardware
- **Processes at the Source**
  - Eliminate ETL bottlenecks
  - Mine data first, govern later

What about this HBase stuff?

- In the beginning, there was the GFS and the MapReduce
  - **And it was Good**
- (Then they realized they needed low latency stuff)
  - And thus BigTable was born, which begat…
- **HBase: BigTable-like storage for Hadoop**
  - Open source
  - Distributed
  - Versioned
  - Column-family oriented
- **HBase** leverages **HDFS** as BigTable leveraged **GFS**

iSAX and Time Series Data

cloudera

What is SAX?

- **S**ymbolic **A**ggregate Appro**X**imation
  - In this case, not XML.
- A symbolic representation of **times series** with some unique properties
  - Essentially a **low pass filter**
  - Lower bounding of Euclidean distance
  - Lower bounding of the DTW distance
  - Dimensionality Reduction
  - Numerosity Reduction

cloudera

SAX in 60 Seconds

- Take time series **T**
- Convert to **Piecewise Aggregate Approximation** (**PAA**)
  - Reduces **dimensionality** of time series **T**
- Then Take **PAA** representation of T and discretize it into a small alphabet of symbols
  - Symbols have a **cardinality** of a, or "number of values they could possibly represent".
- We discretize by taking each **PAA** symbol and finding which horizontally predefined breakpoint the value falls in
  - This gives us the **SAX** letter
  - This complete process converts a time series T into a "SAX word"

cloudera

Why? Time Series Data is “Fuzzy”

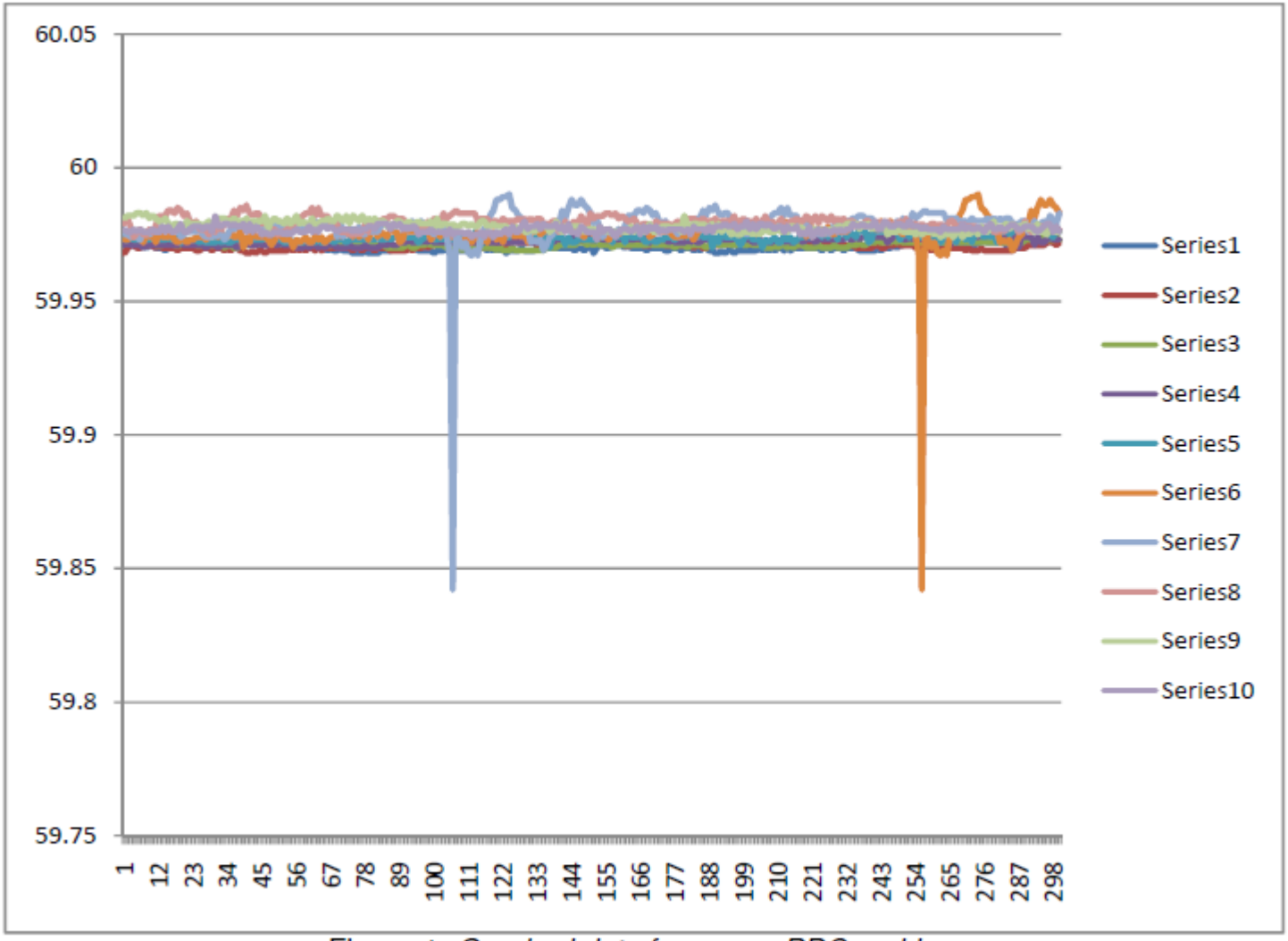
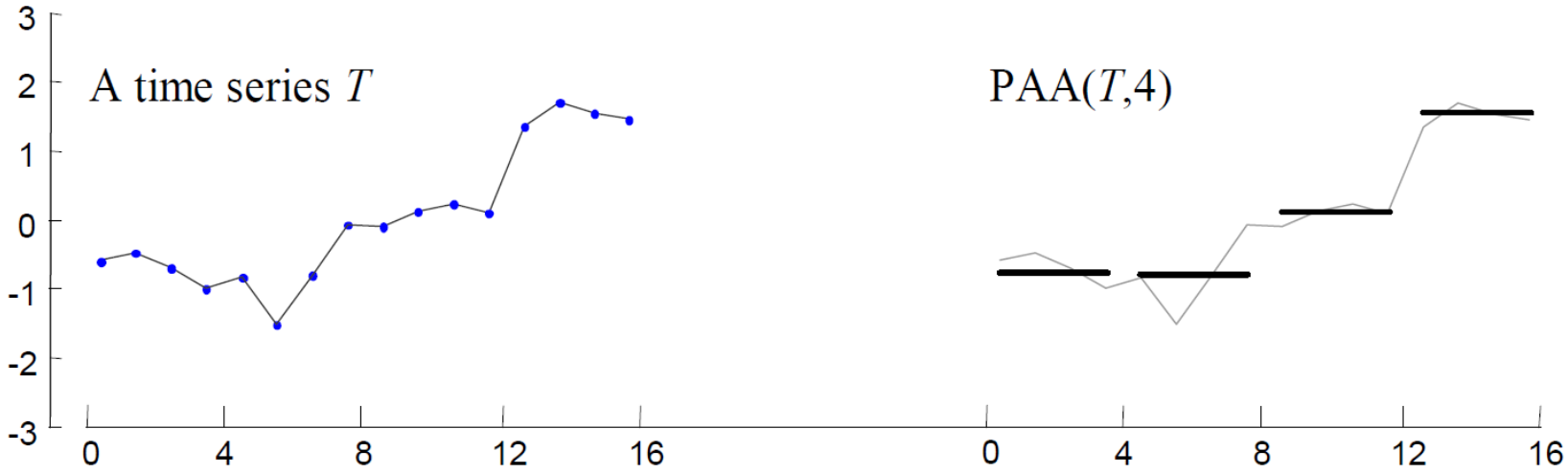


Figure 1. *Graphed data from openPDC archive*

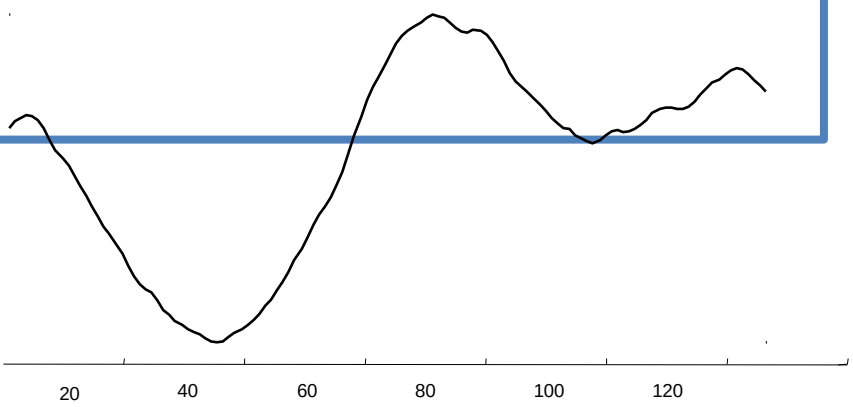
cloudera

SAX: Fuzzy Things Become More Discrete



cloudera

How does SAX work?

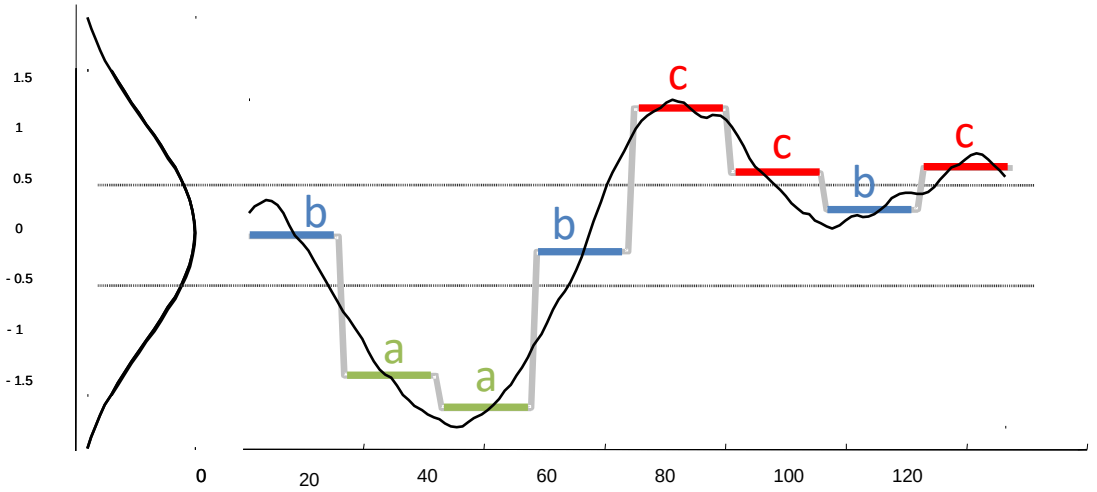


1. First convert the time series to PAA representation

2. Then convert the PAA to symbols to SAX letters:

   baabccbc

(It takes linear time)



Slide Inspired by: http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt

cloudera

SAX and the Potential for Indexing

- The classic SAX representation offers the potential to be indexed
  - If we choose a fixed cardinality of 8, a word length of 4, we end up with 8^4 (4,096) possible buckets to put time series in
  - We could convert a query sequence into a SAX word, and check that "bucket" represented by that word for approximate search as the entries in that bucket are likely very good
    - Exact search would require looking through multiple buckets using lower bounding
  - Data skew presents a problem here
    - Lot's of data in one bucket means lots of scanning!

cloudera

iSAX

- **iSAX**: **i**ndexable **S**ymbolic **A**ggregate appro**X**imation
  - **Modifies SAX** to allow "extensible hashing" and a **multi-resolution representation**
  - Allows for both fast exact search
    - **And ultra fast approximate search**
- **Multi-resolution** property allows us to index time series with zero overlap at leaf nodes
  - Unlike R-trees and other spatial access methods

cloudera

iSAX Word Properties

- Key concepts
  - We can compare iSAX words of different cardinalities
  - We can mix cardinalities per symbol in an iSAX word
- To compare two iSAX words of differing cardinalities
  - We represent each symbol as the bits of its integer
    - Examples
      - Cardinality == 4, “1” as 01, “2” as 10 (4 characters, 0-3 integers)
      - Cardinality == 8, “1” as 001, “2” as 010
  - The trick is, when we “promote” a symbol to a higher cardinality, we add bits to its representation

cloudera

iSAX Dynamic Cardinality

T = *time series 1*

S = *time series 2*

$$iSAX(T,4,8) = \mathbf{T^8} = \{110,110,011,000\} = \{6^8,6^8,3^8,0^8\}$$

$$iSAX(S,4,2) = \mathbf{S^2} = \{0\ \ ,0\ \ ,1\ \ ,1\ \ \} = \{0^2,0^2,1^2,1^2\}$$

$$\{0{**}_1,0{**}_2,1{**}_3,1{**}_4\}$$

$$\mathbf{S^8} = \{011,011,100,100\}$$

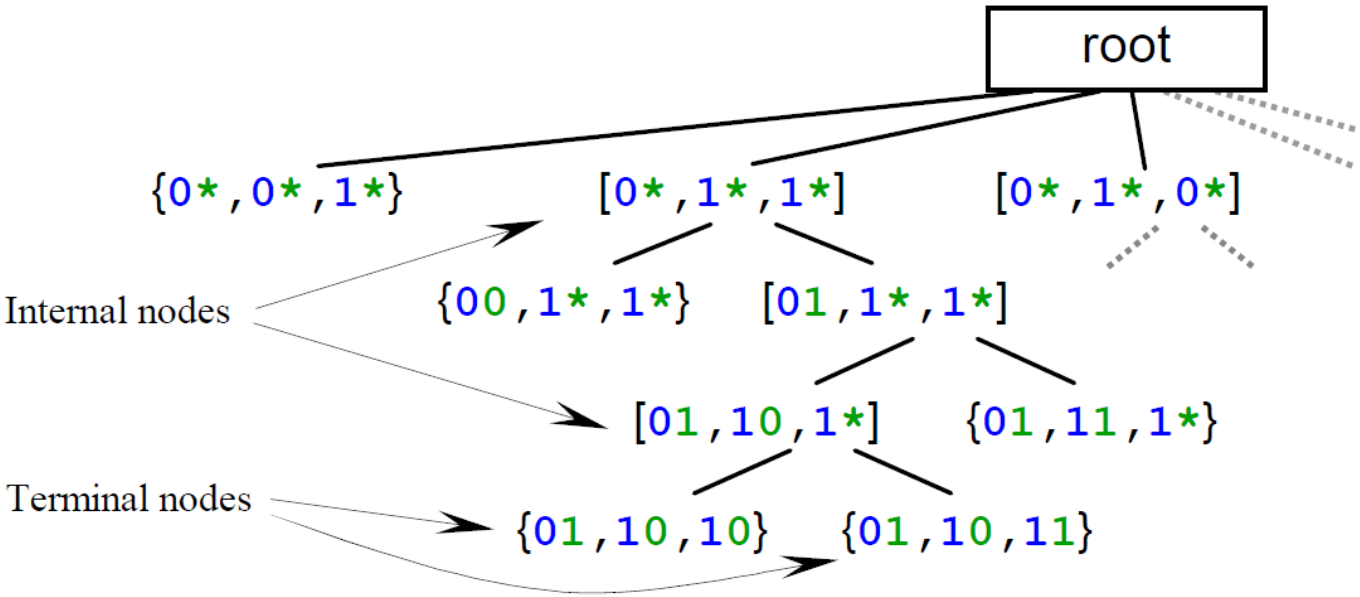
(**S** *fully promoted*)

cloudera

How Does iSAX Indexing Work?

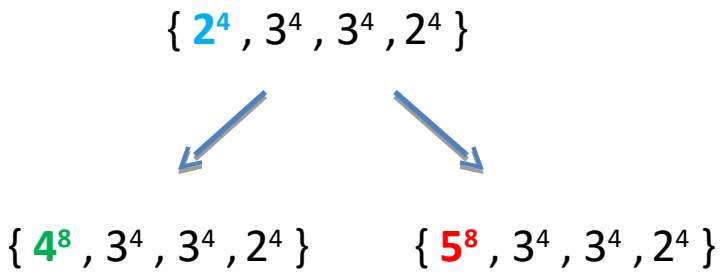
- Similar to a b-tree
  - Nodes represents iSAX words
  - Has internal nodes and leaf nodes
  - Leaf nodes fill up with items to a threshold
  - Once full, a leaf node “splits”
- Each time series sequence to be inserted is converted to a iSAX word
- As we drop down levels in the tree with iSAX-”rehashes”

iSAX Indexing, Inserting, and Searching



Split Mechanics

- *The value "2" at cardinality 4 is "10" in bits*
- *If we split this dimension, we add a bit*
  - *"10" becomes "100" (4) and "101" (5)*
  - *we're now at cardinality of 8*



cloudera

Some Quick Numbers From the iSAX Paper

- 100 million samples indexed, ½ TB of time series data
- Times
  - linear scan: 1800 minutes
  - exact iSAX search 90 minutes
  - **Approx iSAX search: 1.1 sec**
- Quality of results
  - avg rank of NN: 8th
- The bottom line
  - we only have to look @ **0.001%** of the data
  - To find a great quality result in the **top .0001% (NN)**

cloudera

SAX, iSAX, and jmotif

- Both **SAX** and **iSAX** are implemented in the jmotif project
  - Senin Pavel did the original SAX implementation
  - Josh Patterson added the iSAX implementation
  - (Exact search currently is not complete in iSAX)
- Check it out for yourself
  - http://code.google.com/p/jmotif/

What if our Data was … Large?

- And we indexed … a lot of data
  - (And the index got large?)
- iSAX Indexes actually store the sequence sample data
  - This ends up taking potentially a lot of space
- NoSQL gets a bit interesting here
  - Needed fast GETs and PUTs for index nodes
  - **HBase** started to look attractive

cloudera

HBase Strengths

- High write throughput
- Horizontal scalability
- Auto failover
- HDFS Benefits
- With denormalized design, we can lay in any arbitrary computation
  - SQL is not Turing complete

cloudera

Lumberyard: iSAX Indexing and HBase

- Jmotif implements the core logic for iSAX
  - Lumberyard takes this logic, and implements a storage backend
  - **We persist the nodes to rows in Hbase**
- Our potential index size now scales up into the Terabytes
  - We can now leverage Hbase's properties in the storage tier
  - Queries scan a few rows for approximate search
- An ancestor project to Lumberyard was called "**Sparky**"
  - Indexed on content of files in HDFS (sensor / time range)
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/Sparky/

cloudera

Use Cases and Applications

cloudera

How You Would Use iSAX and Lumberyard

- For fast fuzzy query lookups that don't need a perfect best match
  - Just a really good match (fast)
- (We'll fix that exact search thing soon. Promise.)

cloudera

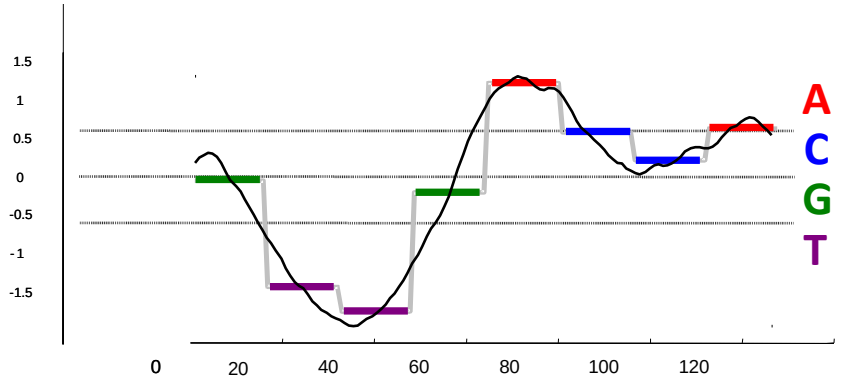
Sensor Data and the openPDC

- Needed to find “unbounded oscillations” in PMU (Smartgrid) Data
- TB’s of data stored in Hadoop
- MapReduce has an interesting property:
  - Sorts numbers really fast via the “shuffle” process
  - Also: my data was not sorted
- Check out the project
  - http://openpdc.codeplex.com/
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- We used SAX and a spatial tree to create a 1NN classifier to detect signal patterns
  - iSAX and Lumberyard allow for faster lookups

cloudera

Genome Data as Time Series

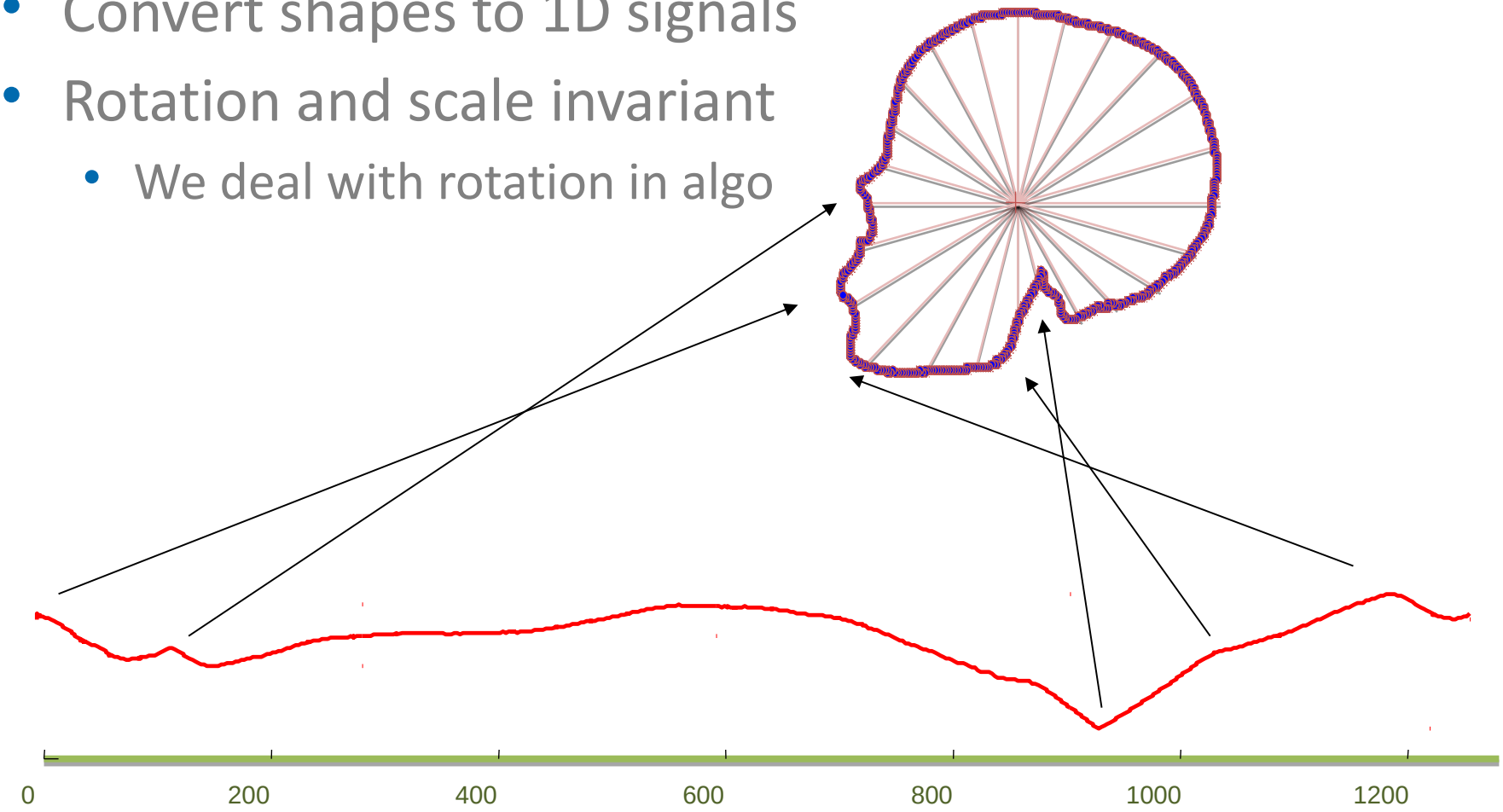
- A, C, G, and T
  - Could be thought of as “1, 2, 3, and 4”!
- If we have sequence X, what is the “closest” subsequence in a genome that is most like it?
  - Doesn’t have to be an exact match!
- Useful in proteomics as well



cloudera

Images…as Time Series?

- Convert shapes to 1D signals
- Rotation and scale invariant
  - We deal with rotation in algo



Slide Inspired by: http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt

cloudera

Other fun Ideas

- Use **Flumebase** and Lumberyard together
  - http://www.flumebase.org/
  - Could provide different query mechanics
- Use **OpenTSDB** with Lumberyard
  - OpenTSB for raw data: http://opentsdb.net/
  - Lumberyard for fuzzy pattern matching
- Image Pattern Matching
  - Imaging could be interesting. Someone needs to write the plugin. (Contact me if interested).
- Use Lumberyard as a 1NN Classifier
  - Check out “Fast time series classification using numerosity reduction” [4]

cloudera

Areas to Improve

- SerDe mechanics
- Finish
  - kNN Search
  - Exact Search
  - MapReduce parallel Index construction
- Lots more testing
- More plugins to decompose data into time series form
  - Image plugin would be interesting
- Performance

cloudera

Try it Yourself at Home

- Download Lumberyard from github
  - https://github.com/jpatanooga/Lumberyard
- Build and Install on Single Machine
  - Build with Ant
  - Setup a few dependencies (Hadoop, Hbase, jmotif)
- Run genomic example: “Genome Data as Time Series”
  - Search for patterns in some sample genome data

cloudera

Lumberyard Summary

- Low latency queries
  - on a lot of time series data
- Experimental, yet has some interesting applications
- Backed by HBase
- Open source ASF 2.0 Licensed

cloudera

(Thank you for your time)

Questions?

Appendix A: Resources

- Lumberyard
  - https://github.com/jpatanooga
- Cloudera
  - http://www.cloudera.com/blog
  - http://wiki.cloudera.com/
  - http://www.cloudera.com/blog/2011/03/simple-moving-average-secondary-sort-and-mapreduce-part-1/
- Hadoop
  - http://hadoop.apache.org/
- HBase
  - http://hbase.apache.org/
- SAX
  - Homepage
    - http://www.cs.ucr.edu/~eamonn/SAX.htm
  - Presentation
    - http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt
- OpenPDC
  - http://openpdc.codeplex.com/
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- The jmotif project provides some support classes for Lumberyard:
  - http://code.google.com/p/jmotif/

cloudera

References

1. SAX
   - http://www.cs.ucr.edu/~eamonn/SAX.htm
2. iSAX
   - http://www.cs.ucr.edu/~eamonn/iSAX.pdf
3. OpenPDC
   - http://openpdc.codeplex.com/
4. Xiaopeng Xi , Eamonn Keogh , Christian Shelton , Li Wei , Chotirat Ann Ratanamahatana, **Fast time series classification using numerosity reduction**, Proceedings of the 23rd international conference on Machine learning, p.1033-1040, June 25-29, 2006, Pittsburgh, Pennsylvania

cloudera

Lumberyard

Time Series Indexing At Scale

Today's speaker – Josh Patterson

- **josh@cloudera.com**
- Master's Thesis: self-organizing mesh networks
  - Published in IAAI-09: TinyTermite: A Secure Routing Algorithm
- Conceived, built, and led Hadoop integration for the openPDC project at TVA
  - Led small team which designed classification techniques for timeseries and Map Reduce
  - Open source work at http://openpdc.codeplex.com
- Now: Sr. Solutions Architect at Cloudera

cloudera

Agenda

- What is Lumberyard?
- A Short History of How We Got Here
- iSAX and Time series Data
- Use Cases and Applications

cloudera

I want to give some context about this project with a short story about hadoop and smartgrid data

Then I want to look deeper into how time series indexing works with iSAX

At the end, I'll bring it all together with Lumberyard and how we might use it

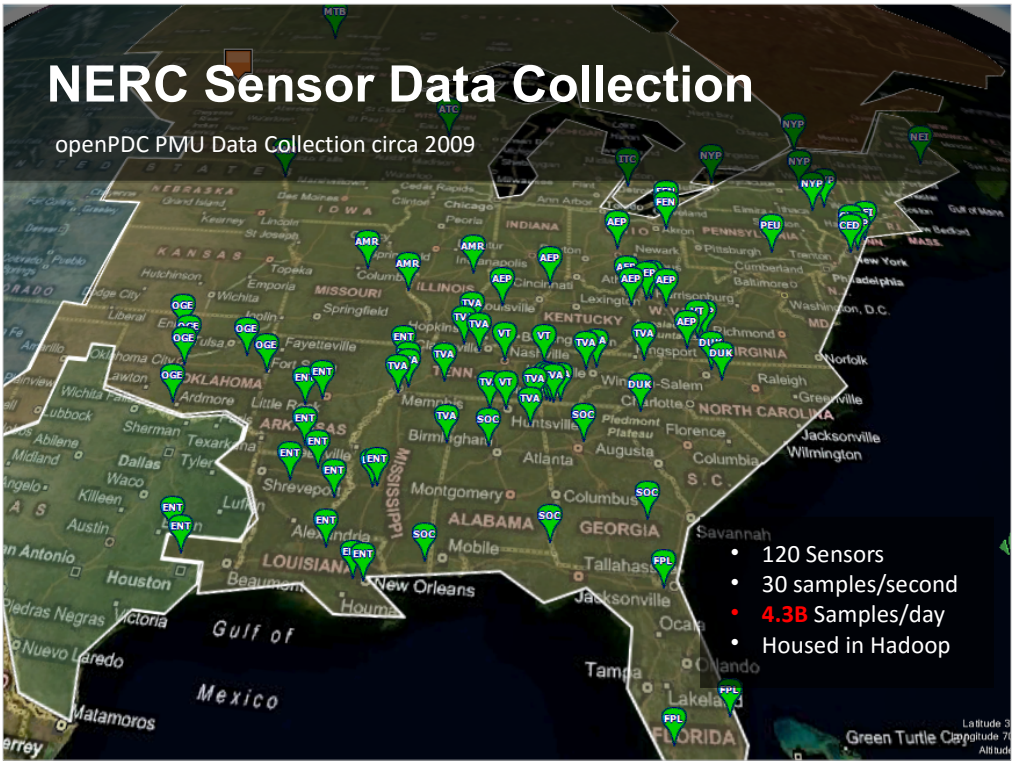
What is Lumberyard?

- **Lumberyard is time series iSAX indexing stored in HBase for persistent and scalable index storage**
- It's interesting for
  - Indexing large amounts of time series data
  - **Low latency** *fuzzy pattern matching queries* on time series data
- Lumberyard is open source and ASF 2.0 Licensed at Github:
  - https://github.com/jpatanooga/Lumberyard/

cloudera

A Short History of How We Got Here

cloudera



Let's set the stage in the context of story, why we were looking at big data for time series.

Story Time: Keogh, SAX, and the openPDC

- NERC wanted high res smart grid data tracked
  - Started openPDC project @ TVA
    - http://openpdc.codeplex.com/
  - We used Hadoop to store and process time series data
    - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- Needed to find “unbounded oscillations”
  - Time series unwieldy to work with at scale
- We found “SAX” by Keogh and his folks for dealing with time series

cloudera

Ok, so how did we get to this point?

Older SCADA systems take 1 data point per 2-4 seconds --- PMUs --- 30 times a sec, 120 PMUs, Growing by 10x factor

What is time series data?

- Time series data is defined as a sequence of data points measured typically at successive times spaced at uniform time intervals
- Examples in finance
  - daily adjusted close price of a stock at the NYSE
- Example in Sensors / Signal Processing / Smart Grid
  - sensor readings on a power grid occurring 30 times a second.
- For more reference on time series data
  - http://www.cloudera.com/blog/2011/03/simple-moving-average-secondary-sort-and-mapreduce-part-1/

cloudera

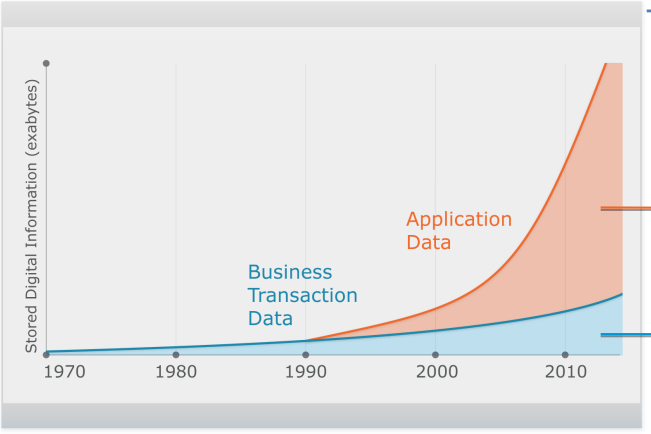
Why Hadoop is Great for the OpenPDC

- Scenario
  - 1 million sensors, collecting sample / 5 min
  - 5 year retention policy
  - Storage needs of 15 TB
- Processing
  - Single Machine: 15TB takes 2.2 DAYS to scan
  - Hadoop @ 20 nodes: **Same task takes 11 Minutes**

cloudera

So, I don't have a smartgrid, how does this apply to me?

Unstructured Data Explosion



(you)

Complex, Unstructured

Relational

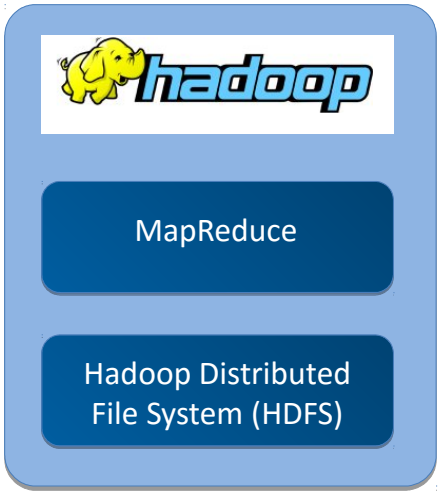
- 2,500 exabytes of new information in 2012 with Internet as primary driver
- Digital universe grew by 62% last year to 800K petabytes and will grow to 1.2 "zettabytes" this year

cloudera

Ben Black wasn't kidding when he said time series was growing fast

Apache Hadoop

Open Source Distributed Storage and Processing Engine



- **Consolidates Mixed Data**
  - Move complex and relational data into a single repository
- **Stores Inexpensively**
  - Keep raw data always available
  - Use industry standard hardware
- **Processes at the Source**
  - Eliminate ETL bottlenecks
  - Mine data first, govern later

cloudera

What about this HBase stuff?

- In the beginning, there was the GFS and the MapReduce
  - **And it was Good**
- (Then they realized they needed low latency stuff)
  - And thus BigTable was born, which begat…
- **HBase: BigTable-like storage for Hadoop**
  - Open source
  - Distributed
  - Versioned
  - Column-family oriented
- **HBase** leverages **HDFS** as BigTable leveraged **GFS**

cloudera

Stealing line from Omer's Hbase deck

---- now recap the story we've told, and tease into "SAX"

iSAX and Time Series Data

cloudera

Ok, so we've set the stage as to why time series data at scale is interesting to me

Now let's talk about some ways to deal with time series data with SAX and iSAX

What is SAX?

- **S**ymbolic **A**ggregate Appro**X**imation
  - In this case, not XML.
- A symbolic representation of **times series** with some unique properties
  - Essentially a **low pass filter**
  - Lower bounding of Euclidean distance
  - Lower bounding of the DTW distance
  - Dimensionality Reduction
  - Numerosity Reduction

cloudera

Whats interesting about symbolic reps?

- Hashing
- Suffix Trees
- Markov Models
- Stealing ideas from text processing/ bioinformatics community
- It allows us to get a better “handle” on time series data
  - Since time series data can be very unwieldy in its “natural habitat”

So, SAX? Whats interesting about that? What is it?

While there are more than 200 different symbolic or discrete ways to approximate time series, none except SAX allows lower bounding

Lower bounding means the estimated distance in the reduced space is always less than or equal to the distance in the original space.

SAX in 60 Seconds

- Take time series **T**
- Convert to **Piecewise Aggregate Approximation** (**PAA**)
  - Reduces **dimensionality** of time series **T**
- Then Take **PAA** representation of T and discretize it into a small alphabet of symbols
  - Symbols have a **cardinality** of a, or "number of values they could possibly represent".
- We discretize by taking each **PAA** symbol and finding which horizontally predefined breakpoint the value falls in
  - This gives us the **SAX** letter
  - This complete process converts a time series T into a "SAX word"

cloudera

Dimensionality in this context means: "number of letters or numbers in the sequence ({1, 2, 3, 4 } **vs** { 1.5, 3.5 })
Example: cardinality 2 == { a, b }

Why? Time Series Data is “Fuzzy”

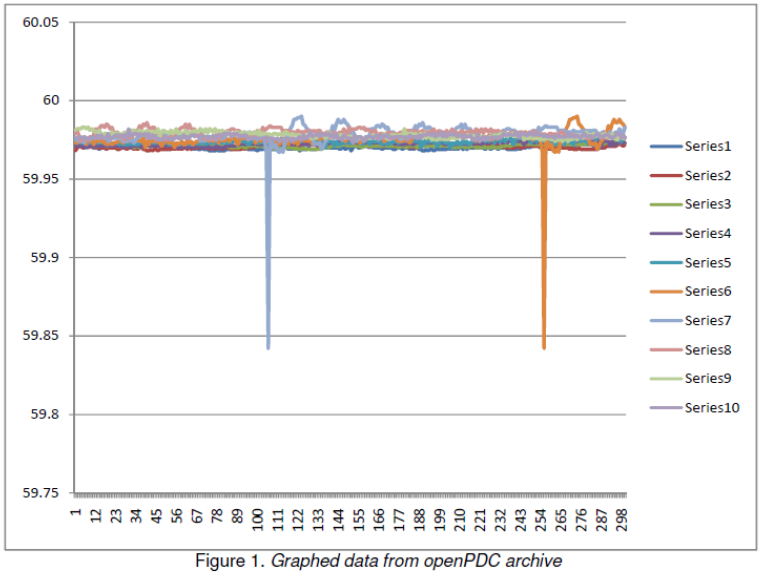
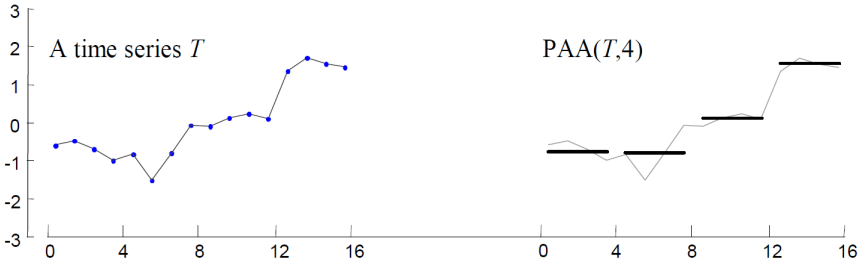


Figure 1. *Graphed data from openPDC archive*

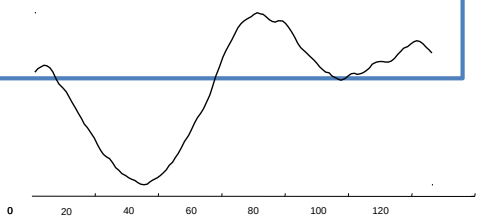
cloudera

SAX: Fuzzy Things Become More Discrete



cloudera

How does SAX work?

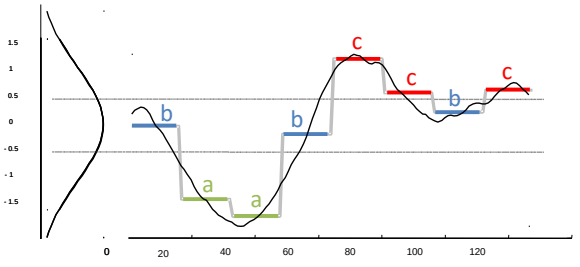


1. First convert the time series to PAA representation

2. Then convert the PAA to symbols to SAX letters:

   baabccbc

(It takes linear time)



Slide Inspired by: http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt

cloudera

We make two parameter choices:

1. "word size", which is number of letters in the SAX "word"
2. "alphabet size" which is the number of letters we use, or cardinality

SAX and the Potential for Indexing

- The classic SAX representation offers the potential to be indexed
  - If we choose a fixed cardinality of 8, a word length of 4, we end up with 8^4 (4,096) possible buckets to put time series in
  - We could convert a query sequence into a SAX word, and check that "bucket" represented by that word for approximate search as the entries in that bucket are likely very good
    - Exact search would require looking through multiple buckets using lower bounding
  - Data skew presents a problem here
    - Lot's of data in one bucket means lots of scanning!

cloudera

iSAX offers a bit aware, quantitized, reduced representation with variable granularity
We could potentially have to search through up to 20% of the data in a single file due to data skew, and/or deal with empty files
- If we have to look at 20% of the data in a lot of queries, we can be at most 5 times faster than complete sequential scan!

iSAX

- **iSAX**: **i**ndexable **S**ymbolic **A**ggregate appro**X**imation
  - **Modifies SAX** to allow "extensible hashing" and a **multi-resolution representation**
  - Allows for both fast exact search
    - **And ultra fast approximate search**
- **Multi-resolution** property allows us to index time series with zero overlap at leaf nodes
  - Unlike R-trees and other spatial access methods

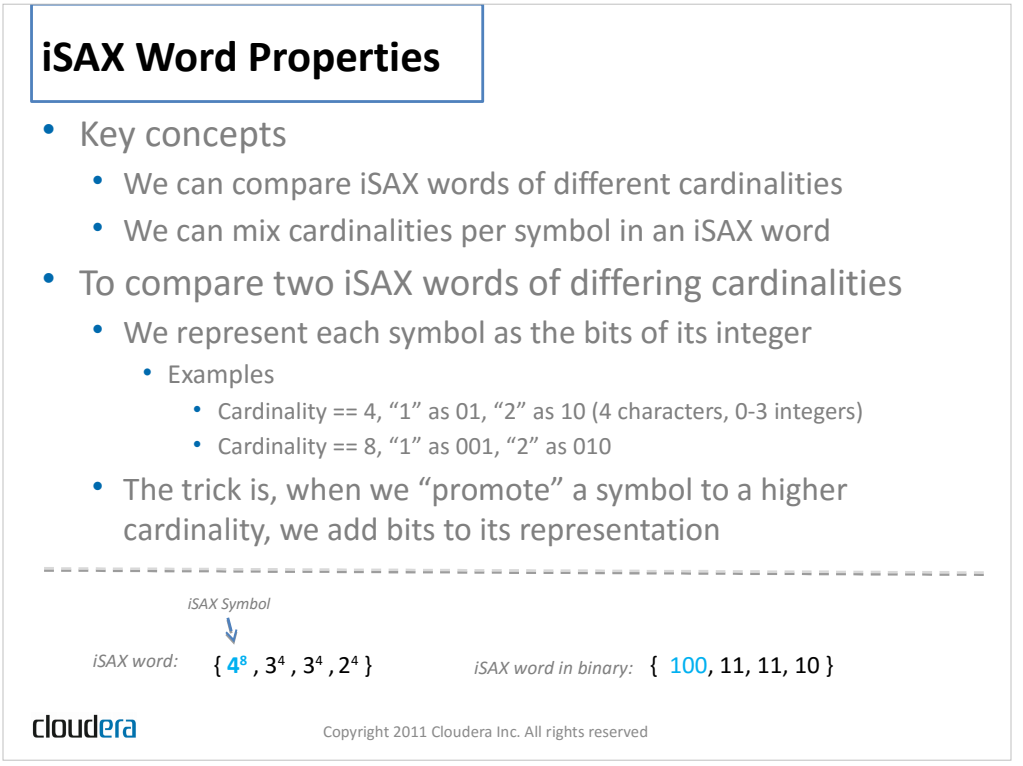
cloudera

iSAX was developed for indexing SAX

allows for a multi resolution representation

similar to extensible hashing

very fast approximate search



A "word" in this context means the letters or symbols representing a time series in SAX / iSAX form

Key concepts (comparison of different cardinalities, mixed cardinalities inside a word)

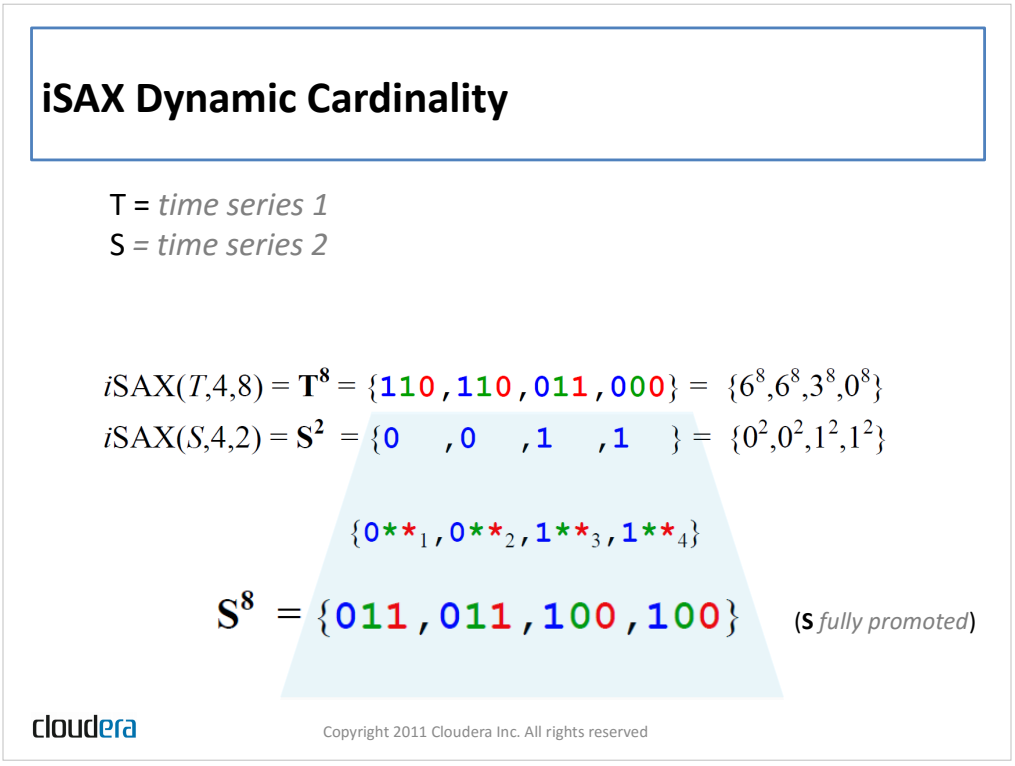
Now each symbol can have its own cardinality

With iSAX, instead of letters we use numbers, and sometimes we use the bit representation of the numbers

Bit representations? What?

When we increase the resolution we add bits to that dimension, or symbol of the word

Individual symbol promotion is a key part of iSAX and directly supports the dynamic cardinality property



If we want to compare two isax words of differing card, we have to promote the lesser to the higher card

Comparison here means a measure of similarity, a distance function

How do we promote cardinality? Remember the part about bit representation?

1. If the firsts bits of S form a prefix, we use the rest of the bits of T
2. If the first bits of S for that position is less than T, we use ‘1’ for the rest of the positions
3. If the first bits of S for that position are greater than T, we use ‘0’ for the rest…

How Does iSAX Indexing Work?

- Similar to a b-tree
  - Nodes represents iSAX words
  - Has internal nodes and leaf nodes
  - Leaf nodes fill up with items to a threshold
  - Once full, a leaf node "splits"
- Each time series sequence to be inserted is converted to a iSAX word
- As we drop down levels in the tree with iSAX-"rehashes"

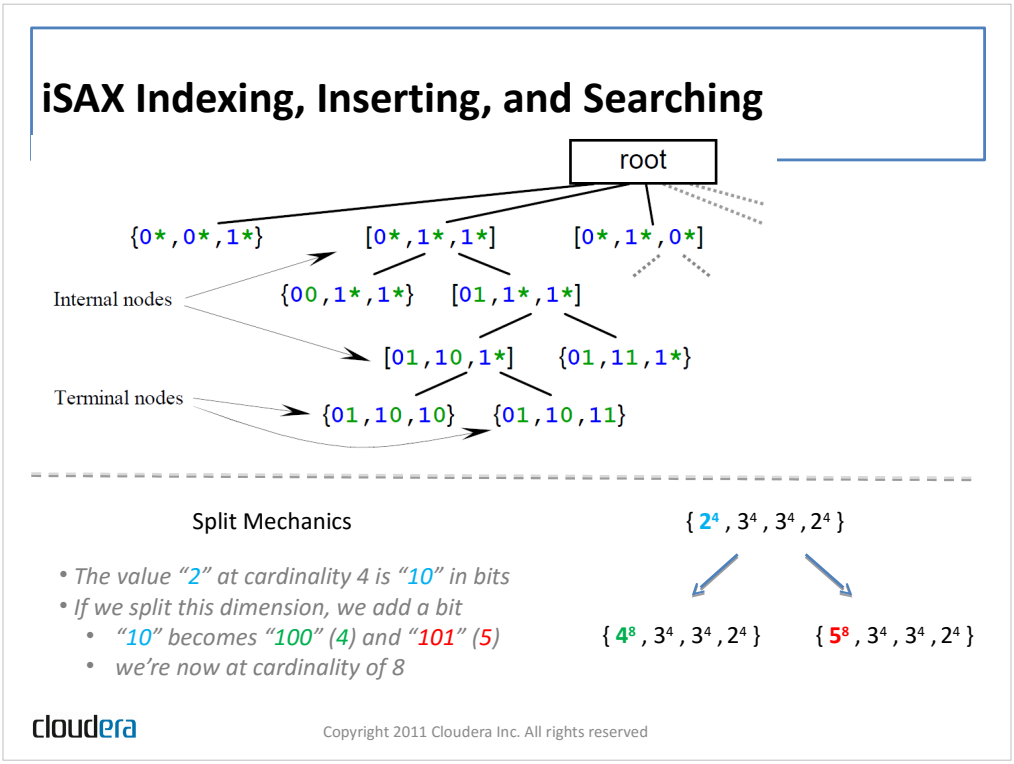
cloudera

Becomes a parent/internal node
Now has two child nodes
All nodes in old leaf node are re-hashed into 1 of the 2 new child nodes

Block2:

At each node as it traverses the tree
Is converted to a iSAX word based on the params of that node

Block3:
Index grows more "bushy" as more items inserted
Our hashes actually gain "resolution" with more information
This property allows us to continually subdivide the search space



Again, takes inspiration from btrees and extensible hashing

The framework converts each time series into an iSAX word before it inserts or searches

Index is constructed with
- Given base card: b
- Word len: w
- Threshold: th

Index hierarchically subdivides the SAX space, dividing regions further as we insert more items

Notice the dimensions being split (on individual symbols in the "words") as we move down the tree

If you'll notice, we can see the split mechanics illustrated near the bottom

Some Quick Numbers From the iSAX Paper

- 100 million samples indexed, ½ TB of time series data
- Times
  - linear scan: 1800 minutes
  - exact iSAX search 90 minutes
  - **Approx iSAX search: 1.1 sec**
- Quality of results
  - avg rank of NN: 8th
- The bottom line
  - we only have to look @ **0.001%** of the data
  - To find a great quality result in the **top .0001% (NN)**

cloudera

Whats perfect accuracy worth to you?

SAX, iSAX, and jmotif

- Both **SAX** and **iSAX** are implemented in the jmotif project
  - Senin Pavel did the original SAX implementation
  - Josh Patterson added the iSAX implementation
  - (Exact search currently is not complete in iSAX)
- Check it out for yourself
  - http://code.google.com/p/jmotif/

cloudera

What if our Data was … Large?

- And we indexed … a lot of data
  - (And the index got large?)
- iSAX Indexes actually store the sequence sample data
  - This ends up taking potentially a lot of space
- NoSQL gets a bit interesting here
  - Needed fast GETs and PUTs for index nodes
  - **HBase** started to look attractive

cloudera

Sparky was a nice experiment, so I knew this Hbase stuff was good

HBase Strengths

- High write throughput
- Horizontal scalability
- Auto failover
- HDFS Benefits
- With denormalized design, we can lay in any arbitrary computation
  - SQL is not Turing complete

cloudera

Lumberyard: iSAX Indexing and HBase

- Jmotif implements the core logic for iSAX
  - Lumberyard takes this logic, and implements a storage backend
  - **We persist the nodes to rows in Hbase**
- Our potential index size now scales up into the Terabytes
  - We can now leverage Hbase's properties in the storage tier
  - Queries scan a few rows for approximate search
- An ancestor project to Lumberyard was called "**Sparky**"
  - Indexed on content of files in HDFS (sensor / time range)
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/Sparky/

cloudera

Contributions of this project are twofold

1. Implementation of the iSAX indexing algorithms in jmotif
2. Addition of Hbase as the backend to iSAX indexes and jmotif

Use Cases and Applications

cloudera

So you are saying “that’s great, but I don’t happen to have a smart grid at home…”

How You Would Use iSAX and Lumberyard

- For fast fuzzy query lookups that don't need a perfect best match
  - Just a really good match (fast)
- (We'll fix that exact search thing soon. Promise.)

cloudera

Sensor Data and the openPDC

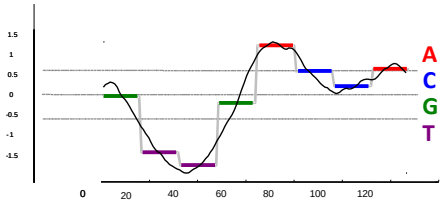
- Needed to find “unbounded oscillations” in PMU (Smartgrid) Data
- TB’s of data stored in Hadoop
- MapReduce has an interesting property:
  - Sorts numbers really fast via the “shuffle” process
  - Also: my data was not sorted
- Check out the project
  - http://openpdc.codeplex.com/
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- We used SAX and a spatial tree to create a 1NN classifier to detect signal patterns
  - iSAX and Lumberyard allow for faster lookups

cloudera

Mention GE and Turbine data

Genome Data as Time Series

- A, C, G, and T
  - Could be thought of as “1, 2, 3, and 4”!
- If we have sequence X, what is the “closest” subsequence in a genome that is most like it?
  - Doesn’t have to be an exact match!
- Useful in proteomics as well



cloudera

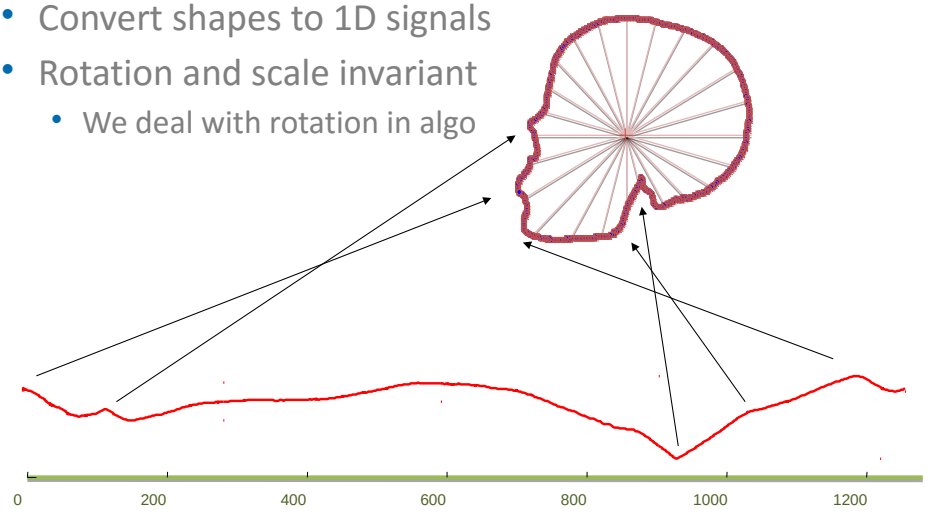
On Monday Steve from google talked about working with genomic data --- genomic data is time series

Our take home demo actually works with a small bit of genomic data

Lots of chatter @ oscon about genomics, I just sat in one today

Images…as Time Series?

- Convert shapes to 1D signals
- Rotation and scale invariant
  - We deal with rotation in algo



Slide Inspired by: http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt

cloudera

Find outline of image (can be tricky, can be done)
Convery shape to 1d signal, time series form
Index with sax
Use various string based tools now

Other fun Ideas

- Use **Flumebase** and Lumberyard together
  - http://www.flumebase.org/
  - Could provide different query mechanics
- Use **OpenTSDB** with Lumberyard
  - OpenTSB for raw data: http://opentsdb.net/
  - Lumberyard for fuzzy pattern matching
- Image Pattern Matching
  - Imaging could be interesting. Someone needs to write the plugin. (Contact me if interested).
- Use Lumberyard as a 1NN Classifier
  - Check out “Fast time series classification using numerosity reduction” [4]

cloudera

Areas to Improve

- SerDe mechanics
- Finish
  - kNN Search
  - Exact Search
  - MapReduce parallel Index construction
- Lots more testing
- More plugins to decompose data into time series form
  - Image plugin would be interesting
- Performance

cloudera

Try it Yourself at Home

- Download Lumberyard from github
  - https://github.com/jpatanooga/Lumberyard
- Build and Install on Single Machine
  - Build with Ant
  - Setup a few dependencies (Hadoop, Hbase, jmotif)
- Run genomic example: “Genome Data as Time Series”
  - Search for patterns in some sample genome data

cloudera

Lumberyard Summary

- Low latency queries
  - on a lot of time series data
- Experimental, yet has some interesting applications
- Backed by HBase
- Open source ASF 2.0 Licensed

cloudera

(Thank you for your time)

Questions?

Appendix A: Resources

- Lumberyard
  - https://github.com/jpatanooga
- Cloudera
  - http://www.cloudera.com/blog
  - http://wiki.cloudera.com/
  - http://www.cloudera.com/blog/2011/03/simple-moving-average-secondary-sort-and-mapreduce-part-1/
- Hadoop
  - http://hadoop.apache.org/
- HBase
  - http://hbase.apache.org/
- SAX
  - Homepage
    - http://www.cs.ucr.edu/~eamonn/SAX.htm
  - Presentation
    - http://www.cs.ucr.edu/~eamonn/SIGKDD_2007.ppt
- OpenPDC
  - http://openpdc.codeplex.com/
  - https://openpdc.svn.codeplex.com/svn/Hadoop/Current%20Version/
- The jmotif project provides some support classes for Lumberyard:
  - http://code.google.com/p/jmotif/

cloudera

References

1. SAX
   - http://www.cs.ucr.edu/~eamonn/SAX.htm
2. iSAX
   - http://www.cs.ucr.edu/~eamonn/iSAX.pdf
3. OpenPDC
   - http://openpdc.codeplex.com/
4. Xiaopeng Xi , Eamonn Keogh , Christian Shelton , Li Wei , Chotirat Ann Ratanamahatana, **Fast time series classification using numerosity reduction**, Proceedings of the 23rd international conference on Machine learning, p.1033-1040, June 25-29, 2006, Pittsburgh, Pennsylvania

cloudera