Pivotal

A NEW PLATFORM FOR A NEW ERA

In-Database Text Analytics

Yu Yang (myang@pivotal.io)

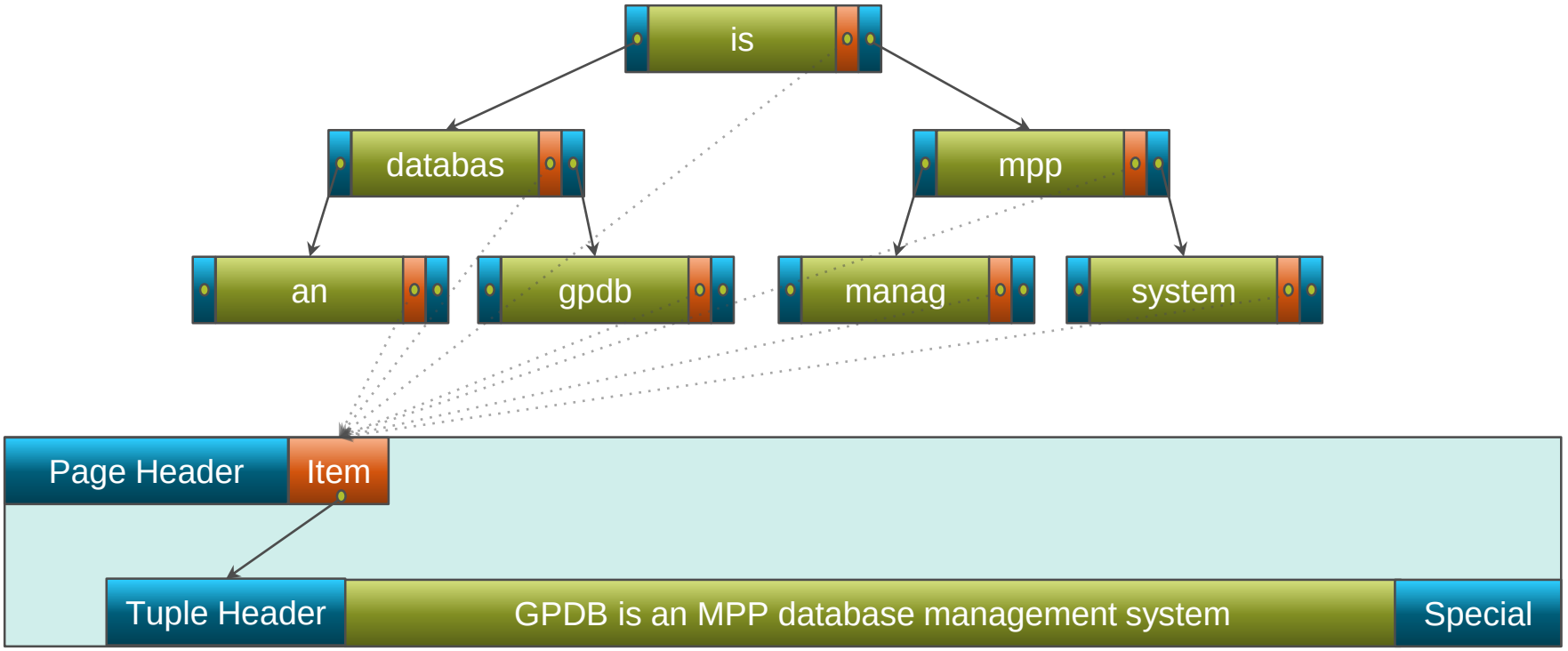
Full-Text Search Principles

Pivotal

Introduction

- Why cannot we simply use an index for full-text search?
  - Consider the phrase ‘GPDB is an MPP database management system’
  - Tokenize - split into array of words: [‘GPDB’, ‘is’, ‘an’, ‘MPP’, ‘database’, ‘management’, ‘system’]::text[]
  - Move to lowcase: [‘gpdb’, ‘is’, ‘an’, ‘mpp’, ‘database’, ‘management’, ‘system’]::text[]
  - Stem using Porter stemming: [‘gpdb’, ‘is’, ‘an’, ‘mpp’, ‘databas’, ‘manag’, ‘system’]::text[]
  - Build binary index

Introduction



- Now what happen if we query “Cool MPP database”?

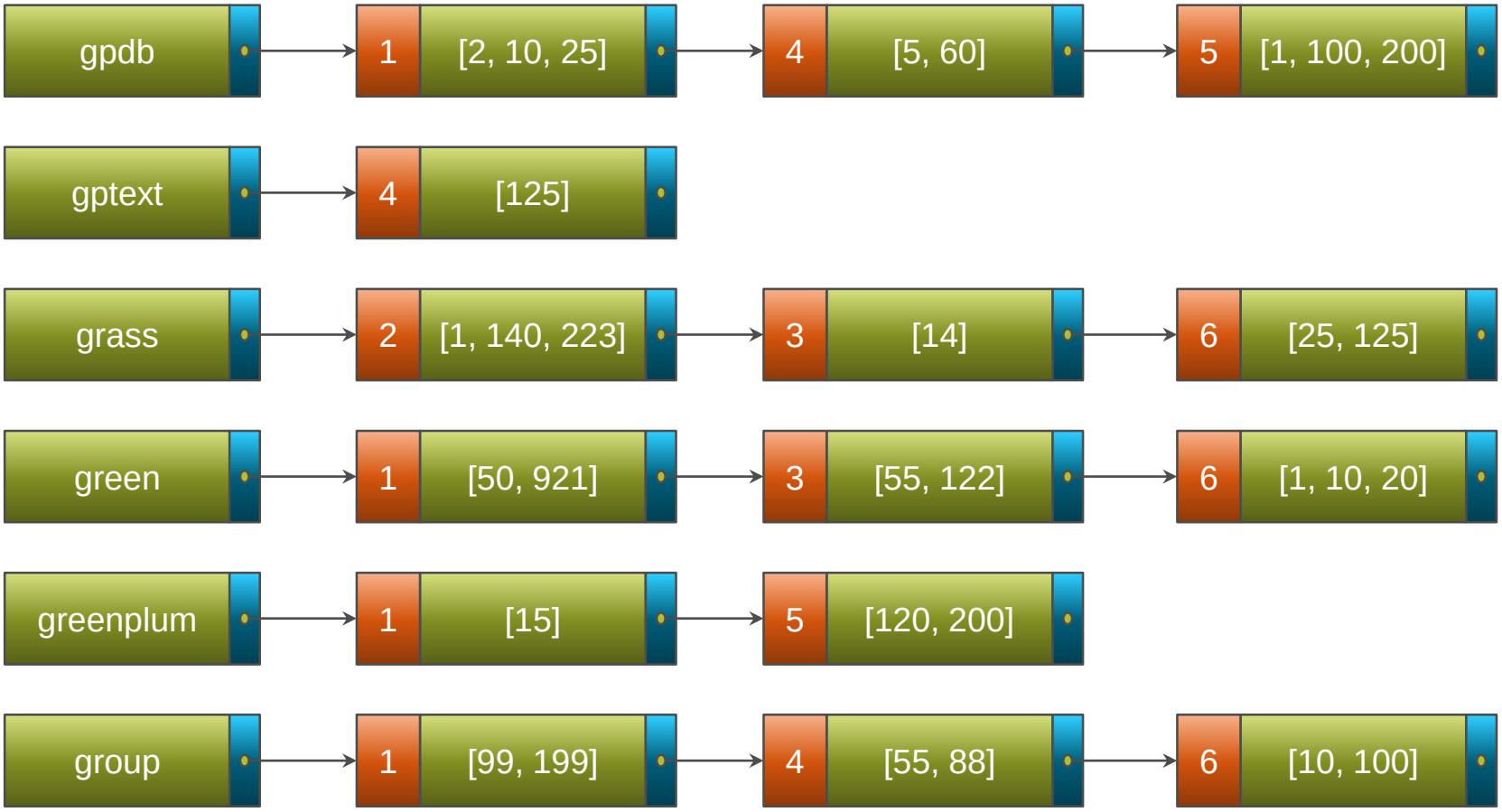
Introduction

- Each term for each document would be stored many times – no way in B-tree or B+-tree to store one key and many links to data
- How to estimate which word is more important and which one is not?
- How to handle complex queries that require exact phrase matching?
- How to rank query results?

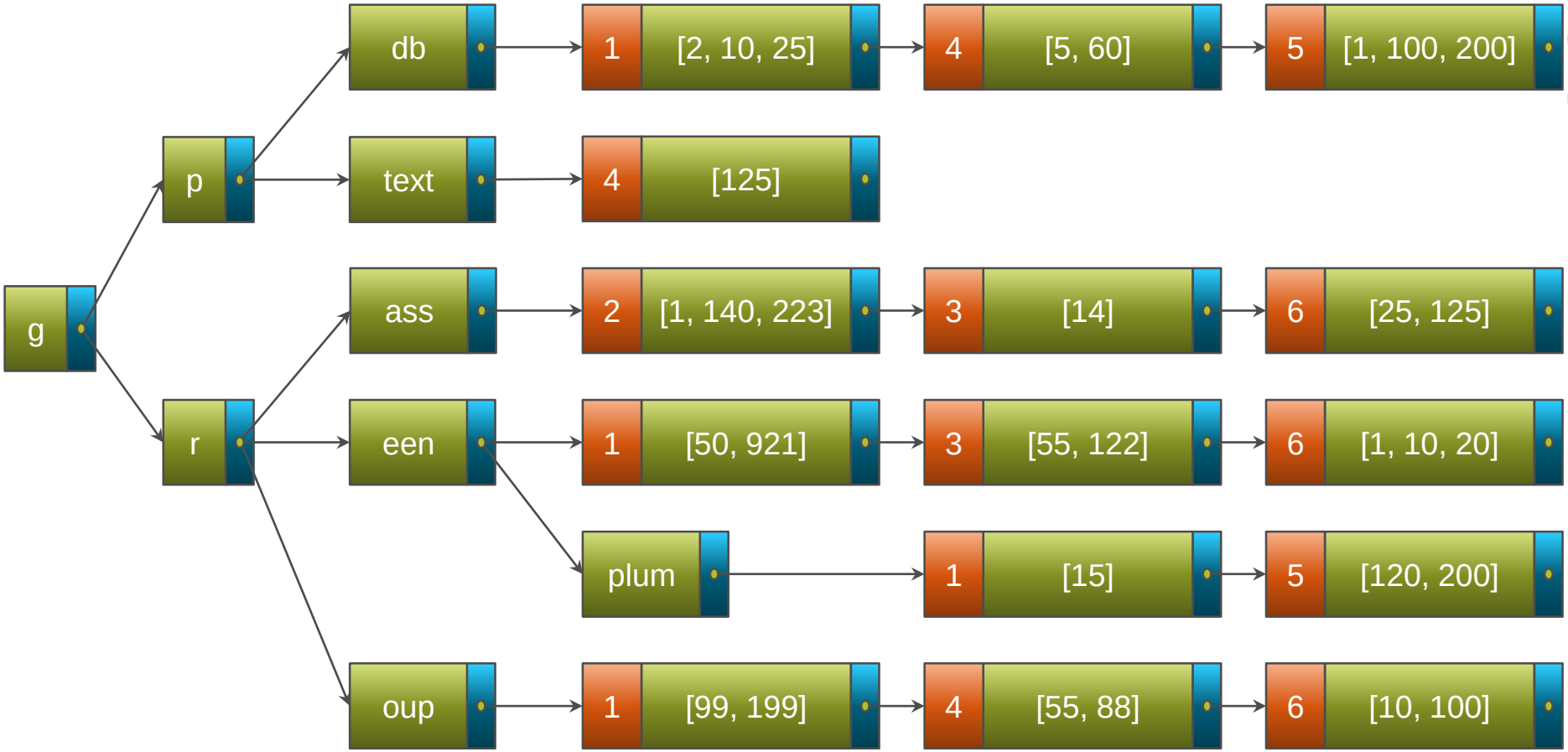
Introduction

- There are 4 main tasks
  - Preprocess data before putting it to index
  - Have an index structure optimized for full-text searches
  - Process incoming queries quickly
  - Rank the results based on how well the query match the document

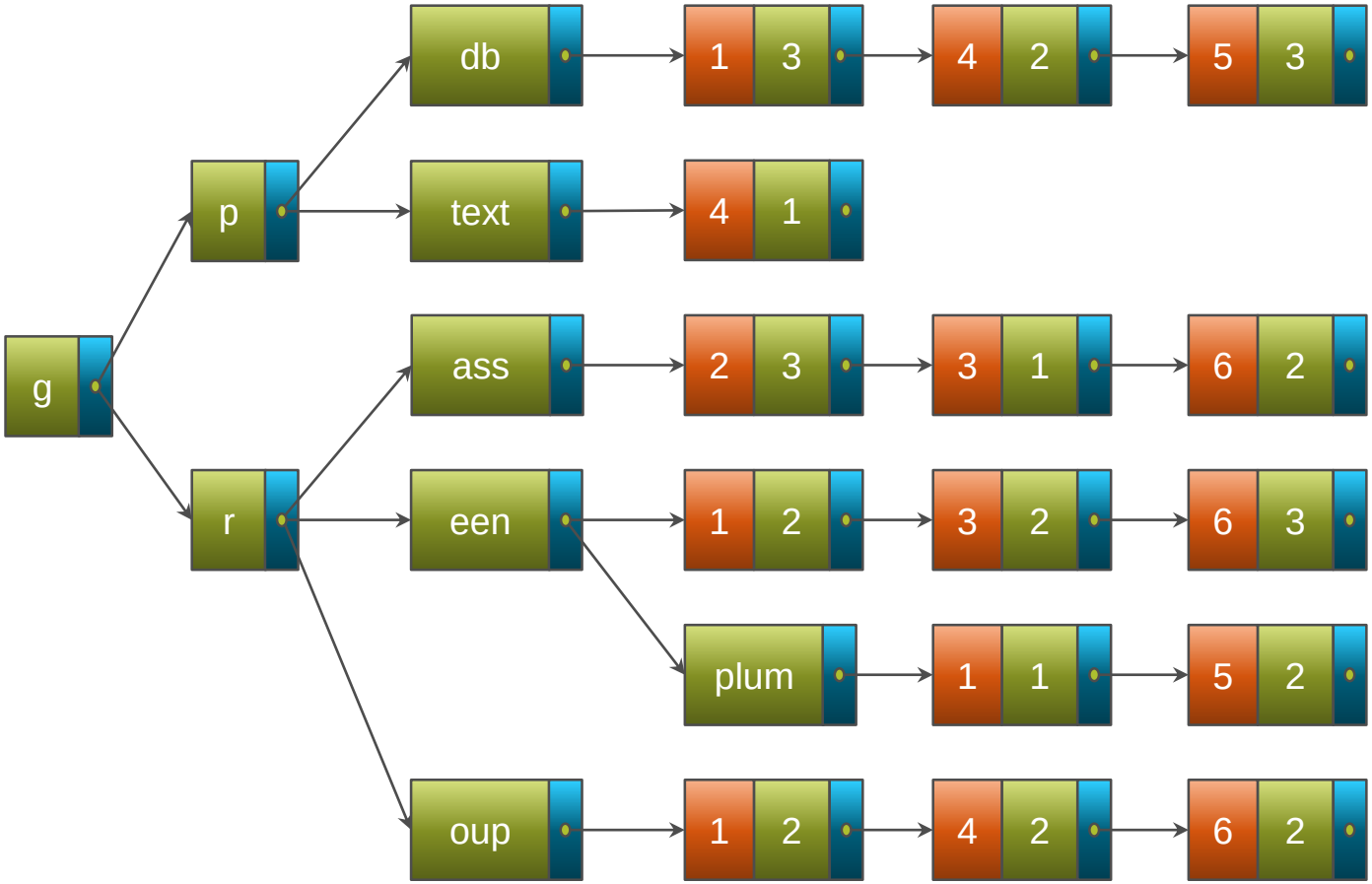
Inverted Index



Inverted Index. Prefix Tree with positions



Inverted Index. Prefix Tree without positions



Example

	Text #			
	1	2	3	4
is	1	1	1	1
a	1	1		
for			1	
in		1		
of			1	
with			1	1
Greenplum	1			2
MPP	1			
database	1	1	1	1
management	1			
system	1			
MADlib		1		
distributed	1		1	
library		1		
analytics		1		
GPText			1	
Solr			1	
functions			1	
interface			1	
Apache			1	1
HD				1
Hadoop				1
tightly				1
integrated				1

→

	TF-IDF			
	1	2	3	4
is	0.00	0.00	0.00	0.00
a	0.04	0.04		
for			0.05	
in		0.09		
of			0.05	
with			0.03	0.03
Greenplum	0.04			0.06
MPP	0.08			
database	0.00	0.00	0.00	0.00
management	0.08			
system	0.08			
MADlib		0.09		
distributed	0.04		0.03	
library		0.09		
analytics		0.09		
GPText			0.05	
Solr			0.05	
functions			0.05	
interface			0.05	
Apache			0.03	0.03
HD				0.06
Hadoop				0.06
tightly				0.06
integrated				0.06

Query “distributed database Greenplum”

	Inverted Index			
	1	2	3	4
is	0.00	0.00	0.00	0.00
a	0.04	0.04		
for			0.05	
in		0.09		
of			0.05	
with			0.03	0.03
Greenplum	0.04			0.06
MPP	0.08			
database	0.00	0.00	0.00	0.00
management	0.08			
system	0.08			
MADlib		0.09		
distributed	0.04		0.03	
library		0.09		
analytics		0.09		
GPText			0.05	
Solr			0.05	
functions			0.05	
interface			0.05	
Apache			0.03	0.03
HD				0.06
Hadoop				0.06
tightly				0.06
integrated				0.06

*

	Query
is	
a	
for	
in	
of	
with	
Greenplum	1.00
MPP	
database	1.00
management	
system	
MADlib	
distributed	1.00
library	
analytics	
GPText	
Solr	
functions	
interface	
Apache	
HD	
Hadoop	
tightly	
integrated	

=

	Documents			
	1	2	3	4
is				
a				
for				
in				
of				
with				
Greenplum	0.04			0.06
MPP				
database	0.00	0.00	0.00	0.00
management				
system				
MADlib				
distributed	0.04		0.03	
library				
analytics				
GPText				
Solr				
functions				
interface				
Apache				
HD				
Hadoop				
tightly				
integrated				

→

Text #	Weigth
1	0.08
2	0.00
3	0.03
4	0.06

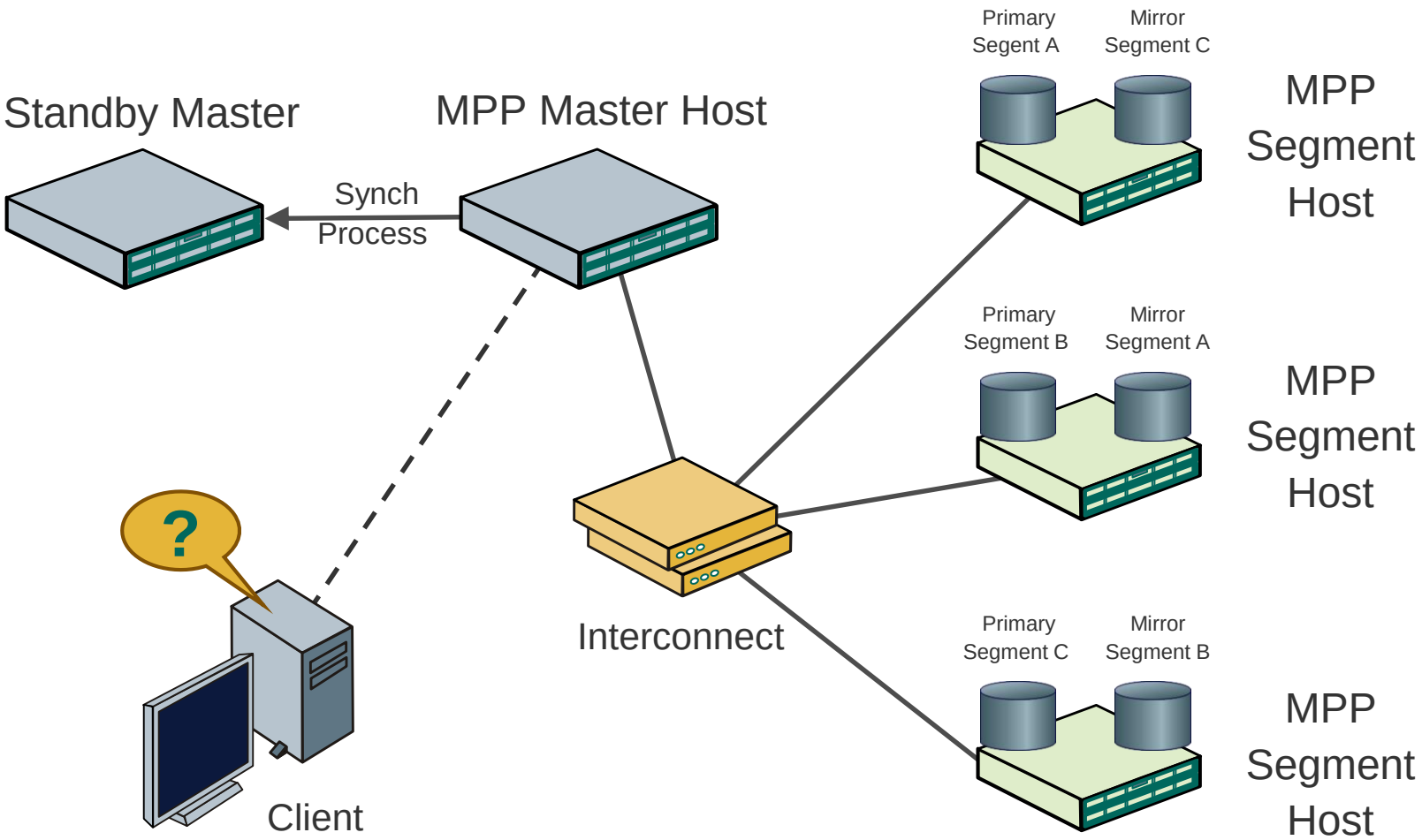
↓

Ranked Result
1
4
3

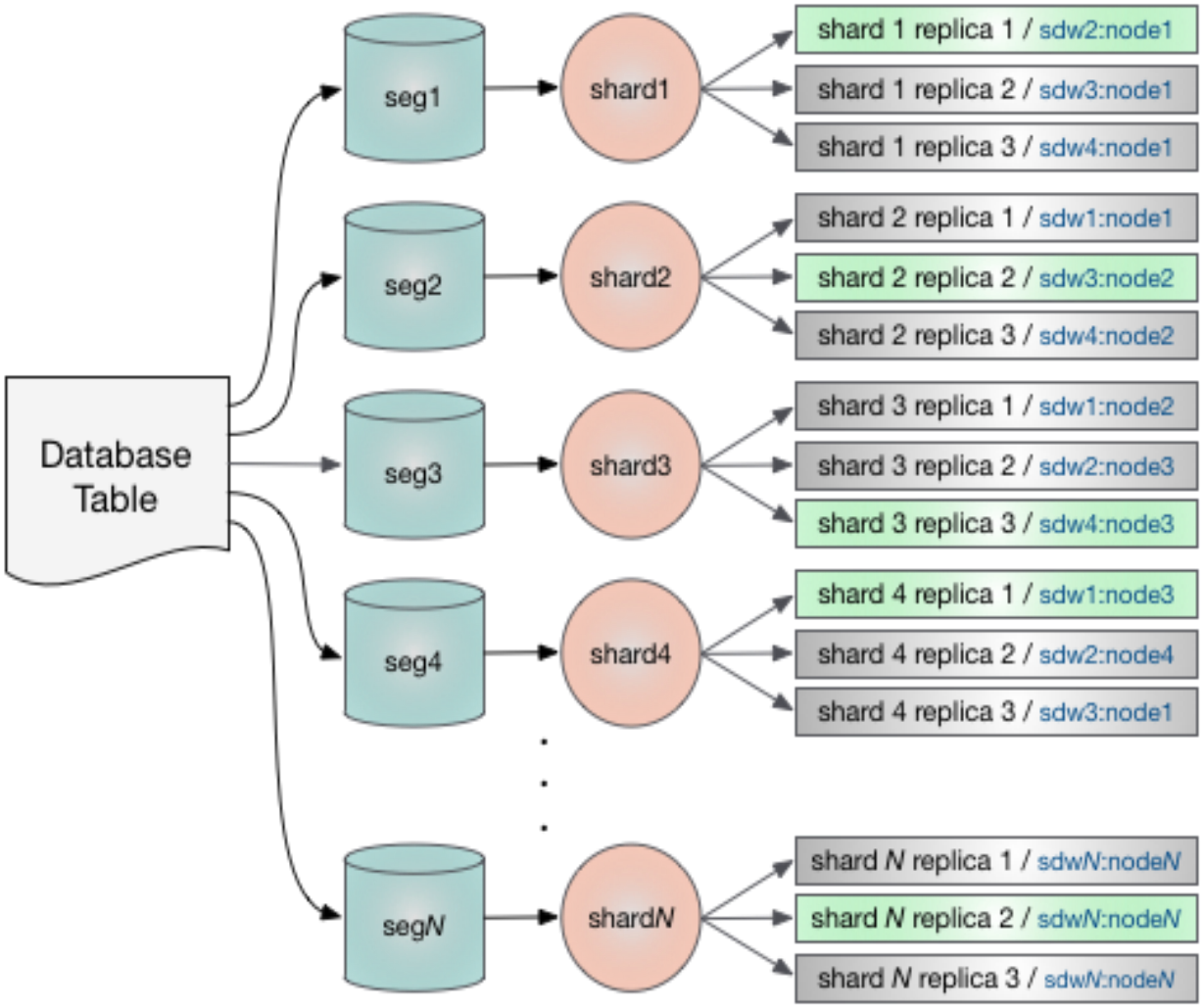
Architecture Review

Pivotal

Greenplum Architecture Review



General Architecture – GPText



Main Functions

- Index functions
  - create index, index, drop index, commit index, etc.
- Search functions
  - search, faceted search, etc.
- Terms operations
  - enable terms for index, create terms table for specific query, document vector creation, etc.
- General functions
  - Index statistics, gptext status, gptext version, etc.

Main Functions

- Create Index
  - Metadata operation in Solr, simple REST API call
- Index Data
  - Function executes on the segment level
  - Function accepts stream of table tuples and passes them to the Solr instance
  - Solr performs analysis of passed in text and adds them to the index
- Commit Index
  - Metadata operation in Solr, simple REST API call

Main Functions

- Search
  - Simple REST API call performed by each of the segments
  - Each of the segments returns data separately and does not know about results returned by other segments
- Drop index
  - Metadata operation in Solr that causes index files to drop, simple REST API call

GPText vs. Solr

- Distributed Solr - Highly scalable
  - Leverages GPDB MPP architecture to distribute nodes and shards across multiple hosts.
- SQL Interface
  - All Solr calls and functionality are exposed through SQL UDFs for a consistent interface through psql
- Ease of Configurability
  - GUC Configuration management
  - gptext-config
- Integrated with GPDB and included in BDS!

GPText vs. Solr (cont.)

- International/Social Media Analyzer Chains
  - Additional Analyzer Chains with custom Tokenizers/Filters to be able to accurately parse out International Text (including CJK characters) and Social Media (hashtags, emoticons, links, etc.)
- Unified Query Parser
  - Combines Solr and Lucene Query parsers into a single unified interface
- Enables term vector support for text analytics with MADLib
  - LDA, K-Means Clustering, SVM

Use Cases

Pivotal

Use Cases

- Financial Email Fraud - Viewing internal emails and analyzing for instances of insider training.
- Medical Records - Diagnostics and Risk of readmittance
- Recommendation Systems
  - Sentiment Analysis
- Anything with Text!

GPText Installation

Pivotal

Requirements

- An operational Greenplum cluster
- Java – JRE > 1.7
- Optional – Additional segment memory

Installation Steps

Run the installation binary from the master:

```
greenplum-text-<version>.bin –c gptext_install_config
gptext_installsql <dbname>
```

- This will install the GPText software alongside GPDB on all nodes in the cluster
- Typically installed into the */usr/local/greenplum-text-<version>* directory
- Source */usr/local/greenplum-text/greenplum-text_path.sh* to set environment

gptext_install_config

- DATA_DIRECTORY: Specifies how many Solr nodes you will create per host and where their data will be located.
- JAVA_OPTS: Specifies how much memory each node will use. Can be changed later with gptext_config
- GPTEXT_PORT_BASE && GP_MAX_PORT_LIMIT
  - Port range for GPText instances.
  - GPText will find available ports in this range.

gptext_install_config (cont.)

- ZOO_CLUSTER
  - BINDING - deploys ZK instances within GPDB cluster
  - If you have an external ZK cluster, you can specify host:port list for this cluster
- ZOO_HOSTS
  - Used with BINDING GPText
  - Specifies how many ZK instances and on what nodes you want them running in your cluster. Should be either 3, 5, or 7.
- ZOO_DATA_DIR
  - Used with BINDING GPText. Sets the ZK data directory
- ZOO_GPTXTNODE
  - Node Path in ZK for GPText. Default (recommended) is gptext
- ZOO_PORT_BASE && ZOO_MAX_PORT_LIMIT
  - Port range for ZK instances.
  - GPText will dynamically find an available port in this range

Uninstalling

Uninstallation binary provided in installation directory

```
gptext-uninstall
```

- This will completely remove gptext and associated components from the cluster

Administration

Pivotal

Administration Utilities

Starting and Stopping:

```
gptext-start
```

```
gptext-stop
```

- Note: Use after GPDB is running

Administration Utilities

Checking status:

```
gptext-state
```

```
SELECT * FROM gptext.status();
```

- Can user either command line utility or database query

Cluster status

Obtain still running index nodes:

```
SELECT * FROM gptext.live_nodes ();
```

Obtain status for index cluster:

```
SELECT * FROM gptext.cluster_status (
);
```

Index Status

Obtain status from one index:

```
SELECT * FROM gptext.index_status (
        ‘<schema>.<table>’
);
```

Obtain status from all indexes:

```
SELECT * FROM gptext.index_status (
);
```

Core Statistics

Obtain statistics from one core:

```
SELECT * FROM gptext.core_statistics (
        '<core_name>'
);
```

Obtain statistics from all cores:

```
SELECT * FROM gptext.core_statistics (
);
```

Index Status & Statistics

Index statistics from the command line:

```
gptext-state --stats
```

```
gptext-state --stats --index <index_name>
```

```
gptext-state --stats --index <index_name> --stats-columns <column>
```

Index Status & Statistics

List all indexes that have been created:

```
gptext-state --list
```

Index Creation

Pivotal

Text Indexes

Creating an index:

```
SELECT * FROM gptext.create_index(
          'schema_name',
          'table_name',
          'id_col_name',
          'default_search_col_name'
);
```

- id_col_name - Name of identifier column, must be a unique bigint
- default_search_col_name – Name of column to search if not specified in query
- Note that all columns in the table will be added to the index by default, unwanted columns must be removed by configuring schema.xml

Text Indexes

Dropping an index:

```
SELECT * FROM gptext.drop_index(
        'index_name'
);
```

- Dropping an index completely removes it from Greenplum and Solr
- This operation can not be undone

Text Indexes

Fully qualified index names:

```
<database>.<schema>.<table>
```

- Fully qualified names must be used to access an index from command line utilities

Data Loading

Pivotal

Data Loading Considerations

Increase max CSV line length:

```
gpconfig -c gp_max_csv_line_length -v 4194304
```

- GPText use cases often require longer text values than might ordinarily be encountered in a database
- Increase from the default of 1MB to the maximum of 4MB
- Can be loaded into Greenplum directly

Data Loading Considerations

Common data types: text

```
1,5.4,some text,more\,text\,data
```

- One row allowed per line
- Special characters inside a value must be escaped
- Quotes not allowed
- Inflexible, but fastest way to load
- Can be loaded into Greenplum directly

Data Loading Considerations

Data Encodings

- For best results all data should be encoded using UTF8

Database Loading Review

External tables with GPFDist:

```
gpfdist -p 8080 -f ”data.csv" -t 30 -m 10485760
```

```
INSERT INTO <table> SELECT * FROM <external_table>;
```

- Fastest Greenplum parallel data loading mechanism

```
gpload -f <data>.yaml
```

Full Text Index

Pivotal

Table Functions

Table Value Expressions:

```
TABLE( SELECT * FROM table SCATTER BY id )
```

- Used as the input to an Enhanced Table Function (ETF)
- Creates a pseudo-table, in parallel, that allows users to pipeline data processing for the purpose of improving query performance
- Resulting data can be ordered or distributed with ‘ORDER BY’ and ‘SCATTER BY’ SQL clauses

Table Functions

Enhanced Table Functions (ETFs):

```
SELECT * FROM my_etf(
        TABLE( SELECT * FROM table SCATTER BY id )
);
```

- Accepts a Table Value Expression as input
- Can output a set of records
- Executes in parallel on data segments
- Planner uses the Table Value Expression to estimate the number of records the ETF will return
- Used by gptext.index() and gptext.search() functions to enhance performance and guarantee parallel execution of data operations

Text Indexing

Adding data to an index:

```
SELECT * FROM gptext.index(
        TABLE( SELECT * FROM <table> ),
        <schema>.<table>
);
```

- Inserts data into a GPText index
- First parameter can be an entire table (SELECT *), or you can selectively index by changing the inner select list, for example: add a where clause, remove columns, add columns from another table, etc.
- Be sure the inner select has the same distribution policy as the table you are indexing. Use ‘SCATTER BY’ to achieve this:

```
TABLE( SELECT * FROM <table> SCATTER BY <distrib_id> )
```

Text Indexing

Committing an index:

```
SELECT * FROM gptext.commit_index( ‘<database>.<schema>.<table>’ );
```

- Data added to an index must be committed before it will be available to users
- You can add data to an index multiple times before committing

Text Indexing

Rolling back an index operation:

```
SELECT * FROM gptext.rollback_index( ‘<database>.<schema>.<table>’ );
```

- In the case of a mistake or failure, you can rollback an index operation
- Will undo any index operations performed since the last time the index was committed
- This includes delete operations!

Text Indexing

Deleting from an index:

```
SELECT * FROM gptext.delete(
        ‘<database>.<schema>.<table>’,
        <query>
);
```

- Deletes some or all data from an index
- The query ‘<*:*>’ would effectively truncate an index
- The query ‘iphone’ would only delete documents which match that query
- Must be committed in the same way as an index load

Full Text Queries

Pivotal

Text Search Queries

Basic Query:

```
SELECT * FROM gptext.search(
        table_value_expression,
        index_name,
        search_query,
        filter_queries,
        [options]
);
```

- Searches text index for <search_query> and returns id's and relevance scores of matching documents
- <table_value_expression> used to ensure parallel execution
- <search_query> uses enhanced Solr/Lucene query parser syntax

Text Search Queries

Basic Query Example:

```
SELECT * FROM gptext.search(
        TABLE( SELECT 1 SCATTER BY 1),
        myschema.mytable,
        “cat AND dog”,
        NULL,
        NULL
);
```

Text Search Queries

Joining result with original table:

```
SELECT t.id t.title, q.score FROM
        mytable t,
        SELECT * FROM gptext.search(
                TABLE( SELECT 1 SCATTER BY 1),
                myschema.mytable,
                “cat AND dog”,
                NULL,
                NULL
        ) q
        WHERE t.id = q.id
);
```

- Allows you to return more than just id and relevancy score
- Query can have arbitrary complexity - join with additional tables or functions

Text Search Queries

Top 10 results:

```
SELECT t.id t.title, q.score FROM
          mytable t,
          SELECT * FROM gptext.search(
                    TABLE( SELECT 1 SCATTER BY 1),
                    myschema.mytable,
                    “cat AND dog”,
                    NULL,
                    ‘rows=10’
          ) q
          WHERE t.id = q.id
          ORDER BY q.score DESC LIMIT 10
);
```

- Uses <options> search function parameter
- Returns top 10 results sorted by Solr relevancy score

Text Search Queries

Search Count:

```
SELECT * FROM gptext.search_count(
        myschema.mytable,
        'cat AND dog',
        'userid:25'
);
```

- Used when you only care about the number of documents that match
- Much faster than a search because no results are returned from the indexes

Faceted Search

Pivotal

Text Search Queries

Faceted Field Search:

```
SELECT * FROM gptext.faceted_field_search(
          myschema.mytable,
          search_query,
          filter_query,
          facet_fields,
          facet_limit,
          minimum
);
```

- Aggregates or performs a ‘groups by’ on the results
- Provides a count of matching documents for each distinct ‘facet_field’ in the result
- facet_limit: maximum number of facets to return
- minimum: only return values with more than ‘minimum’ matching documents

Text Search Queries

Faceted Field Search Example:

```
SELECT * FROM gptext.faceted_field_search(
          myschema.mytable,
          'cat OR dog',
          'country:USA',
          '{state}',
          0,
          1
);
```

- Return the number of documents that match the query 'cat OR dog', grouped by the state the documents came from
- Only return states that have more than 1 matching document
- Result: (California, 25), (Florida, 15), (Idaho, 3), etc…

Text Search Queries

Faceted Query Search Example:

```
SELECT * FROM gptext.faceted_query_search(
        myschema.mytable,
        'cat OR dog',
        'country:USA',
        '{price:[* TO 100], price:[101 TO 200], price:[201 TO 300], price:[301 TO *]}',
        0,
        1
);
```

- Specify facets as queries instead of field values

Text Search Queries

Faceted Range Search Example:

```
SELECT * FROM gptext.faceted_range_search(
        myschema.mytable,
        'cat OR dog',
        'country:USA',
        'date',
        'NOW/YEAR-1YEAR',
        'NOW/YEAR',
        '+1DAY'
);
```

- Facet by a range of values
- In this example, return a facet for each day for the last 1 year

Query Performance

Pivotal

Query Performance

Improve join performance with an index

```
SELECT t.id t.title, q.score FROM
        mytable t,
        SELECT * FROM gptext.search( … ) q
        WHERE t.id = q.id
);
```

- In this example the join condition is ‘t.id = q.id’
- For many rows this can be a costly operation
- Improve performance by creating a database index on ‘t.id’

Query Performance

Building the index:

```
CREATE INDEX table_id_idx ON table.message USING btree (id);
```

- Creates a btree index on the ‘id’ column of our base table
- Greenplum can use this during search result join to improve performance

Query Performance

Query time parameters:

```
set enable_nestloop=on;
set enable_hashjoin=off;
set enable_seqscan=off;

SELECT t.id t.title, q.score FROM
          mytable t,
          SELECT * FROM gptext.search( … ) q
          WHERE t.id = q.id
);
```

- Set these three parameters at run time to force a nested loop index join
- Can also be set at user level if desired
- Greatly improves execution time of query!

Query Performance

Verify index usage via explain plan:

```
Gather Motion 2:1  (slice3; segments: 2)  (cost=0.00..201.73 rows=11 width=82)
   ->  Nested Loop  (cost=0.00..201.73 rows=11 width=82)
         ->  Redistribute Motion 2:2  (slice2; segments: 2)  (cost=0.00..0.04 rows=1 width=8)
               Hash Key: q.id
               ->  Table Function Scan on search q  (cost=0.00..0.02 rows=1 width=8)
                     ->  Redistribute Motion 1:2  (slice1; segments: 1)  (cost=0.00..0.01
rows=1 width=0)
                           Hash Key: "?column?"
                           ->  Result  (cost=0.00..0.01 rows=1 width=0)
         ->  Index Scan using id_idx on message t  (cost=0.00..201.67 rows=1 width=90)
               Index Cond: t.id = q.id
```

- We can see an Index Scan and a Nested Loop join
- This query will be very performant

Query Performance

See no index usage via explain plan:

```
Gather Motion 2:1  (slice3; segments: 2)  (cost=0.05..36888.50 rows=11 width=82)
   ->  Hash Join  (cost=0.05..36888.50 rows=11 width=82)
         Hash Cond: t.id = q.id
         ->  Seq Scan on message t  (cost=0.00..33334.19 rows=710810 width=90)
         ->  Hash  (cost=0.04..0.04 rows=1 width=8)
               ->  Redistribute Motion 2:2  (slice2; segments: 2)  (cost=0.00..0.04 rows=1
width=8)
                     Hash Key: q.id
                     ->  Table Function Scan on search q  (cost=0.00..0.02 rows=1 width=8)
                           ->  Redistribute Motion 1:2  (slice1; segments: 1)
(cost=0.00..0.01 rows=1 width=0)
                                 Hash Key: "?column?"
                                 ->  Result  (cost=0.00..0.01 rows=1 width=0)
```

- The same query with a different plan
- A sequential scan on the complete base table followed by a hash join
- This query will be much slower as the base table ‘message’ grows in size

Analytics

Pivotal

MADlib

- Open-source library for scalable in-data analytics
- Data-parallel implementation of mathematical, statistical and machine-learning methods
- http://madlib.net

MADlib

Installing MADlib to your database:

```
/usr/local/madlib/bin/madpack -p greenplum -c gpadmin@localhost/demo
install
```

- Like GPText, installs MADlib schema and all necessary database objects

Text Analytics

KMeans example:

```
SELECT * FROM madlib.kmeans_plusplus(
        ‘<tfidf_table>’,
        ‘<tfidf_column’>,
        ‘<document_id’>,
        … MADLib Kmeans paramters …
);
```

- Executes KMeans clustering algorithm from MADlib
- Executes in parallel inside the database
- See MADlib documentation for complete parameter list
- Many other MADlib functions follow a similar model

Text Analytics

Using gptext-analytics:

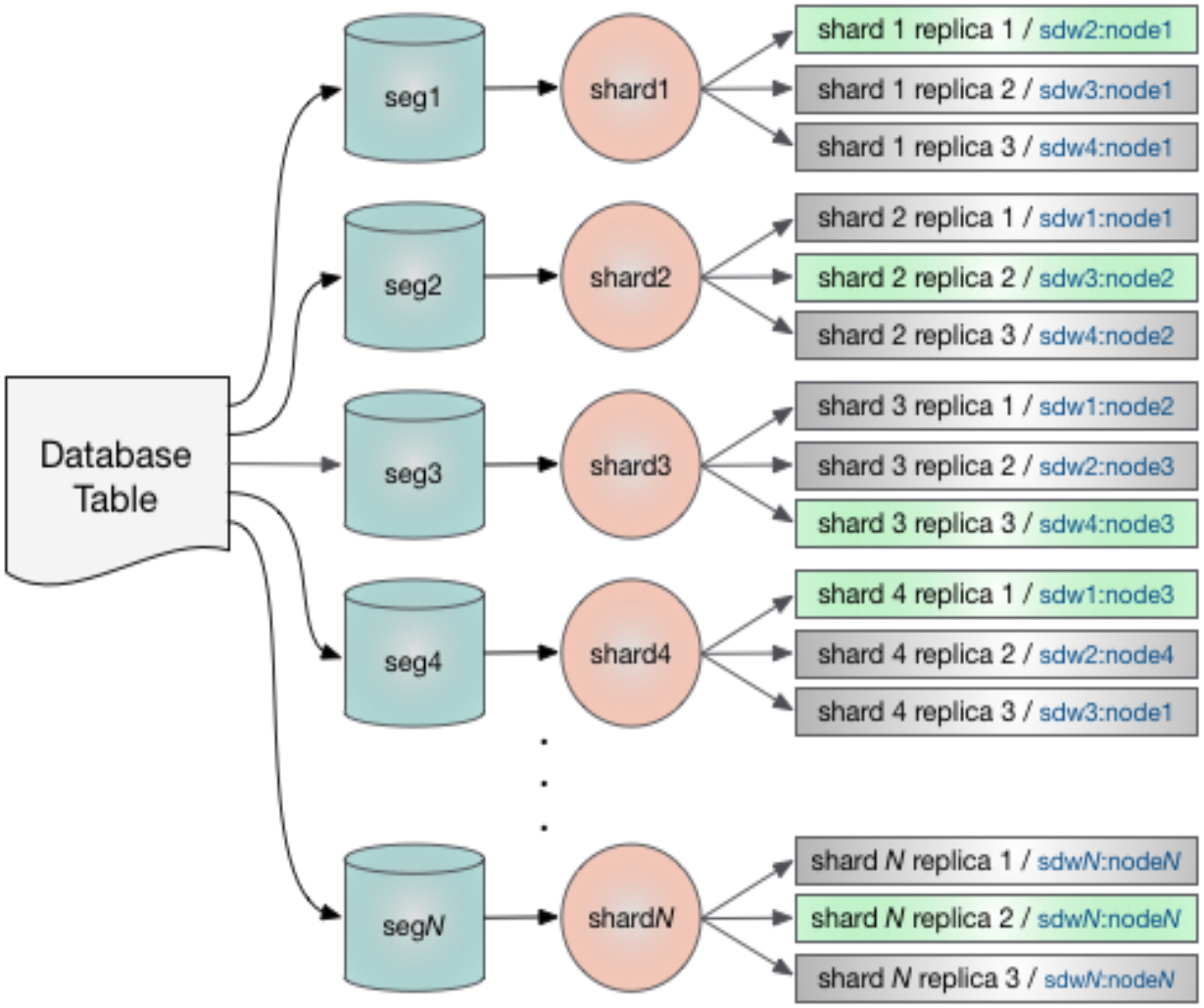
```
$ gptext-analytics -c kmeans -f kmeans.yaml
# Edit auto-generated yaml
$ gptext-analytics -d <directory> -f kmeans.yaml
$ psql -f execute.sql
```

- Helper utility that executes all intermediate steps
- Creates dictionary, term vectors, tf-idf vectors, etc.
- Outputs only kmeans clustering results
- Currently supports: kmeans, svm_train, svm_test
- Over time will be extended to support others

HA with SolrCloud

Pivotal

GPText 2.0 - Let's revisit...



Zookeeper

- Helps manage the overall structure so that both indexing and search requests can be routed properly.
- Keeps configurations in central location.
- Fault tolerant and Highly Available - Runs multiple instances.
- Minimal resource usage.

Shard vs. Replica

- Shard - The act of splitting a single Solr index across multiple machines.
  - Used to distribute data across multiple machines/servers.
  - Scalable.
- Replica - A copy of a given shard.
  - Consists of a Leader and Followers.

Replica - Leader vs. Follower

- Leader distributes documents to be indexed to all other replicas (followers) in the shard.
- Reports that all followers have confirmed receiving a given document.
- When follower receives a document, it adds it to its transaction log and sends a response to the leader.
- If ZK detects a leader has gone down, it will initiate the leader election process instantaneously, selecting a new leader to begin distributing documents again.

zkManager

- Checks Zookeeper cluster state. If ZK was installed with GPText, this can start/stop the ZK cluster
- All gptext utils also check zkManager state. If the ZK cluster is unhealthy, it will issue a warning.

GPText Recover

- Basic Recovery
  - Find all the Solr nodes that are not running.
  - Attempts to recover Solr nodes and any replicas that exist on those nodes.
  - Copies over any data that is not synced to the recovered replicas.
- Force Recovery
  - Deletes the node and recreates it.
  - Recreates replicas that were on the original node.

GPText Recover - New Host

- If a host machine goes down
  - As long as one replica for each shard is up, GPText is operational
  - -H: Specify new hosts to recover nodes from previously down hosts.
- Allows you to stay fully operational in critical situations

gptext-replica

- Add or delete a replica for an index shard.
- -i: Specifies the index name to add/drop the replica
- -s: The name of the shard to add/drop replicas
- -r: Used during drop. The name of the replica to drop. Found from gptext.index_status()

Optional:

- -n: The node to add the replica to. Otherwise, randomly chosen.
- -o: Used for drop. Drops replica *only* if it's down.

GPText configs

Pivotal

gptext-config

- Edit, add, or upload configuration files in Zookeeper.
- Revert configured files in Zookeeper.
- Edit JVM configuration options.
- Upload jar files to the GPText home directory.
- Some changes require re-indexing. gptext-config will execute this if necessary.

gptext-config - Options

- -i: Name of Index to configure.
- -f: File to configure
  - solrconfig.xml, managed-schema, stopwords|synonyms|protwords|emoticons.txt, etc.
- -a: Append to a filename. ‘-f’ and ‘-i’ arguments required.
- -r: Revert to previous version of file.
- -b: Batch Size. # of Solr instances to configure concurrently.
- -d: Distribute an existing file to Solr instances.
- -j: Uploads custom JAR file.
- -o: Modifies JVM options. Important for Memory usage!
- Some changes require re-indexing. gptext-config will execute this if necessary.

managed-schema

- Allows you to change the properties of your index
- type
  - Which Analyzer Chain to use for each field
- stored
  - Whether the original field value should be stored
  - Useful for return full field queries
- indexed
  - Whether a column should be indexed or not during indexing step.

solrconfig.xml

- query
  - filterCache: What filter cache used to improve performance
  - queryResultCache: Caches results of searches
  - documentCache: Caches Lucene Document objects
  - fieldValueCache: Cache used to hold field values that are quickly accessible by document id
  - enableLazyFieldLoading: Stored fields loaded lazily
- requestHandler
  - search handler for /select request
  - update handler for /update request
- queryResponseWriter
  - writer response in given format

GPText GUCs - The important ones!

- replication_factor - The number of replicas per shard for a newly created index.
- failover_factor - Minimum ratio of Solr nodes that must be up in order to create a new index.
- extension_factor - Maximum number of replicas that can be added for an index per GPText node after the index is created.
- search_batch_size - Batch size for search requests. Performance vs. Resource tradeoff
- terms_batch_size - Batch size for terms operations.
- ……

A NEW PLATFORM FOR A NEW ERA

Thanks!