Item 76
git_comments:

1.
2.
3.

git_commits:

1. **summary:** SOLR-2592: refactor routers to separate public classes
   message: SOLR-2592: refactor routers to separate public classes git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/trunk@1418043 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Custom Hashing
   description: If the data in a cloud can be partitioned on some criteria (say range, hash, attribute value etc) It will be easy to narrow down the search to a smaller subset of shards and in effect can achieve more efficient search.

jira_issues_comments:

1. This is why I created the issue SOLR-1431 . It may have a configuration as follows {code:xml} <requestHandler name="standard" class="solr.SearchHandler" default="true"> <!-- other params go here --> <shardHandler class="CloudShardHandler"/> </requestHandler> {code} The CloudShardHandler should lookup ZK and return all the shards return all the shards by default. I should be able to write a custom FqFilterCloudShardHandler and narrow down the requests to one or more shards {code:xml} <requestHandler name="standard" class="solr.SearchHandler" default="true"> <!-- other params go here --> <shardHandler class="FqFilterCloudShardHandler"/> </requestHandler> {code}
2. This patch is intended to be a cocktail napkin sketch to get feedback (as such forwarding queries to the appropriate shards is not yet implemented). I can iterate on this as needed. The attached patch is a very

simple implementation of pluggable sharding which works as follows: 1. Configure a ShardingStrategy in SolrConfig under config/shardingStrategy, if none is configured the default implementation of sharding on the document's unique id will be performed. {code:xml} <shardingStrategy class="solr.UniqueIdShardingStrategy"/> {code} 2. The ShardingStrategy accepts an AddUpdateCommand, DeleteUpdateCommand, or SolrParams to return a BytesRef that is hashed to determine the destination slice. 3. I have only implemented updates at this time, queries are still distributed across all shards in the collection. I have added a param to common.params.ShardParams for a 'shard.keys' parameter that would contain the value(s) which is(are) to be hashed to determine the shard(s) which is(are) to be queried within the the HttpShardHandler.checkDistributed method. if 'shard.keys' does not have a value the query would be distributed across all shards in the collection. Notes: There are no unit tests yet however all existing tests pass. I am not quite sure about the configuration location within solr config, however as sharding is used by both update and search requests placing it in the udpateHandler and (potentially multiple) requestHandler sections would require a duplication of the same information in the solr config for what I believe is more of a collection-wide configuration. As hashing currently requires the lucene.util.BytesRef class the solrj client can not currently hash the request to send the request to a specific node without having solrj add a dependency on lucene core - something that is most likely not desired. Additionally, hashing on a unique id also requires access to the schema as well to determine the field that contains the unique id. Are there any thoughts on how to alter the hashing to remove these dependencies and allow for solrj to be a 'smart' client that submits requests directly to nodes that contain the data? How would solrj work when multiple updates are included in the request that belong to different shards? Send the request to one of the nodes and let the server distribute them to the proper nodes? Perform concurrent requests to the specific nodes?
3. I've been tinkering with this a bit more and discovered that the real-time get component requires the ability to hash on the unique id of a document to determine the shard to forward the request to, requiring any hashing implementation to support hashing based off the unique id of a document. To account for this any custom hashing based on an arbitrary document property or other value would have to hash to the same value as the unique document id. Looking at what representation of the unique id to hash on, by hashing based off of the string value rather than the indexed value, solrj and any other smart client can hash it and submit requests directly to the proper shard. The method oas.common.util.ByteUtils.UTF16toUTF8 would serve the purpose of generating the bytes for the murmur hash on both client and server. On both the client and server side updates and deletes by id are simple enough to hash, queries would require an additional parameter to specify a value to be hashed to determine the shard to forward the request to. It would also be a good idea to be able to optionally specify a value to hash on to direct the query to a particular shard rather than broadcast to all shards. I'm going to rework what I had in the previous patch to account for this.
4. Here is an update to my original patch that accounts for the requirement of hashing based on unique id and works as follows: 1. Configure a ShardKeyParserFactory in SolrConfig under config/shardKeyParserFactory. If there is not one configured the default implementation of sharding on the document's unique id will be performed. The default configuration is equivalent to: {code:xml} <shardKeyParserFactory class="solr.ShardKeyParserFactory"/> {code} 2. The ShardKeyParser has two methods to parse a shard key out of the unique id or a delete by query. The default implementation returns the string value of the unique id when parsing the unique id to forward it to the specific shard, and null when parsing the delete by query to broadcast a delete by query to the entire collection. 3. Queries can be directed to a subset of shards in the collection by specifying one or more shard keys in the request parameter 'shard.keys'. Notes: There are no distinct unit tests for this change yet, however all current unit tests pass. The switch to hashing on the string value rather than the indexed value is how I realized the real-time get component requires support for hashing based on the document's unique id with a failing test. By hashing on the string values rather than indexed values, the solrj client can direct queries to a specific shard however this is not yet implemented. I put the hashing function in the oas.common.cloud.HashPartioner class, which encapsulates the hashing and partitioning in one place. I can see a desire for a pluggable collection partitioning where a collection could be partitioned on time periods or some other criteria but that is outside of the scope of pluggable shard hashing.
5. What's the status on this and does the patch work with the latest trunk version of https://issues.apache.org/jira/browse/SOLR-2358 ? Nick
6. The pluggable_sharding_V2.patch does work with the latest version of trunk, however I would recommend running the unit tests after applying the patch to verify things on your end. I am hesitant to recommend the patch for production use as it does change the value that is hashed from the indexed value to the string value of the unique id which changes the behavior of the current hashing if your unique id is a numeric field type. I am using the patch with the latest from a few days ago for testing and am using composite unique ids (123_456_789) so I created a ShardKeyParser that hashes on the first id in the composite (123) and parses a clause out of a delete by query to route updates and deletes to the appropriate shards. In order to direct the query to a particular shard use the shard.keys param set to the value that should be hashed (123).
7. **body:** Here is an updated patch that fixes an issue where a delete by query is silently dropped. I have not yet added any additional unit tests for this patch, however all existing unit test pass - which may not be that

significant as that did not catch the DBQ issue.
label: test

8. Hash-based distribution (deterministic pseudo-randomness) is only one way to divvy up documents among shards. Another common one is date-based: each month is in a different shard. You can search recent or older without searching all shards. If you're going to make pluggable distribution policies, why stick to hashing? Why not do the general case?
9. {quote} If you're going to make pluggable distribution policies, why stick to hashing? Why not do the general case? {quote} Very good point. The patches I've attached do not attempt to address other partitioning schemes such as date based or federated as my immediate requirement is to be able to customize the hashing. Implementation of a pluggable distribution policy is the logical end state, however I have not had the time to investigate potential approaches for such a change yet.
10. **body:** I have tried out Michael's patch and would like to provide some feedback to the community. We are using a very-recent build from the 4x branch but I grabbed this patch from trunk and tried it out anyway... Our needs were driven by the fact that, currently, the counts returned when using field collapse are only accurate when the documents getting collapsed together are all on the same shard (see comments for https://issues.apache.org/jira/browse/SOLR-2066). For our case we collapse on a field, xyz, so we need to ensure that all documents with the same value for xyz are on the same shard (overall distribution is not a problem here) if we want counting to work. I grabbed the latest patch (dbq_fix.patch) in hopes of finding a solution to our problem. The great news is that Michael's patch worked like a charm for what we needed -- thank you kindly, Michael, for this effort! The not-so-good news is that for our particular issue we needed a way to get at data other than the uniqueKey (the only data available with ShardKeyParser) -- in our case we need access to the xyz field data. Since this implementation provides nothing but uniqueKey we had to encode the xyz data in our uniqueKey (e.g. newUniqueKey = what-used-to-be-our-uniqueKey + xyz), which is certainly less-than-ideal and adds unsavory coupling. Nonetheless, as a fix to a last-minute gotcha (our counts with field collapse need to be accurate in a multi-shard environment) I was happily surprised at how easy it was to find a solution to our particular problem with this patch. I would definitely like to see a second iteration that incorporates the ability to get at other document data, then you could do whatever you want by looking at dates and other fields, etc. though I understand that that probably goes quite a bit deeper in the codebase, especially with distributed search.
label: code-design
11. The reason for requiring the unique id to be hashable is that it is required to support the distributed real-time get component to retrieve a document based on only the unique id, which in turn is required for SolrCloud. Unit tests that exercise the patch thoroughly are still needed and I will be diving into later this week, so please keep that in mind if you are using this outside of a test environment.
12. Attached is an updated patch that includes unit tests and applies cleanly to the latest in branch_4x. This patch corrects a bug found in unit testing where a delete by ID may be not be forwarded to the correct shard. I created a shard key parser implementation for testing that uses the string value of the unique id reversed, and modified the FullSolrCloudDistribCmdsTest test to use that implementation (which is how I found the delete by id bug). I've also added a shard key parser with unit tests that parses the shard key value from an underscore-delimited unique id and from a clause in a delete query. As the shard must be parsed from the unique id for the realtime get handler a composite id of some sort will be required for placing a document in a specific shard. While the solrj client is able to parse the shard key from the query and forward it to the appropriate shard, I have not yet implemented that piece yet.
13. **body:** I've been using some code very similar to Michael's latest patch for a few weeks now and am liking it less and less for our use case. As I described above, we are using this patch to ensure that all docs with the same value for a specific field end up on the same shard -- this is so that the field collapse counting will work for distributed searches, otherwise the returned counts are only an upper bound. For us the counts returned from a search using field collapse have to be exact since that drives paging logic (we can't have a user going to page 29 and finding nothing there). The problems we've encountered have entirely to do with our need to update the value of the field we're doing a field-collapse on. Our approach -- conceptually similar to the CompositeIdShardKeyParserFactory in Michael's latest patch -- involved creating a new schema field, indexId, that was a combination of what used to be our uniqueKey plus the field that we collapse on: *Original schema* {code:xml} <field name="id" type="string" indexed="true" stored="true" multiValued="false" required="true" /> <field name="xyz" type="string" indexed="true" stored="true" multiValued="false" required="true" /> ... <uniqueKey>id</uniqueKey> {code} *Modified schema* {code:xml} <field name="id" type="string" indexed="true" stored="true" multiValued="false" required="true" /> <field name="xyz" type="string" indexed="true" stored="true" multiValued="false" required="true" /> <field name="indexId" type="string" indexed="true" stored="true" multiValued="false" required="true" /> ... <uniqueKey>indexId</uniqueKey> {code} During indexing we insert the extra {{indexId}} data in the form: {{id:xyz}}. Our custom ShardKeyParser extracts out the {{xyz}} portion of the uniqueKey and returns that as the hash value for shard selection. Everything works great in terms of field

collapse, counts, etc. Consider what happens when we need to change the value of the field, xyz, however. Suppose that our document starts out with these values for the 3 fields above: {quote} id=123 xyz=456 indexId=123:456 {quote} We then want to change xyz to the value 789, say. In other words, we want to end up... {quote} id=123 xyz=789 indexId=123:789 {quote} ...so that the doc lives on the same shard along with other docs that have xyz=789. Before any of this we would simply pass in a new document and all would be good since we weren't changing the uniqueKey. However, now we need to delete the old document (with the old uniqueKey) or we'll end up with duplicates. We don't know whether a given update changes the value of xyz or not and we don't know what the old value for xyz was (without doing an additional lookup) so we must include an extra delete along with every change: *Before* {code:xml} <add> <doc> <field name="id">123</field> <field name="xyz">789</field> <doc> </add> {code} *Now* {code:xml} <delete> <query>id:123 AND NOT xyz:789</query> </delete> <add> <doc> <field name="id">123</field> <field name="xyz">789</field> <field name="indexId">123:789</field> <-- old value was 123:456 <doc> </add> {code} So in addition to the "unsavory coupling" between id and xyz there is a significant performance hit to this approach (as we're doing this in the context of NRT). The fundamental issue, of course, is that we only have the uniqueKey value (id) and score for the first phase of distributed search -- we really need the other field that we are using for shard ownership, too. One idea is to have another standard schema field similar to uniqueKey that is used for the purposes of shard distribution: {code:xml} <uniqueKey>id</uniqueKey> <shardKey>xyz</shardKey> {code} Then, as standard procedure, the first phase of distributed search would ask for uniqueKey, shardKey and score. Perhaps the ShardKeyParser gets both uniqueKey and shardKey data for more flexibility. In addition to solving our issue with field collapse counts, date-based sharding could be done by setting the shardKey to a date field and doing appropriate slicing in the ShardKeyParser.
label: code-design

14. **body:** One of the use cases I have is identical to yours Andy, where shard membership is used to ensure accuracy of the numGroups in the response for a distributed grouping query. The challenge in that use case is that during an update a document could potentially move from one shard to another, requiring deleting it from its current shard along with adding it to the shard where it will now reside. If the previous value of the shardKey is not known, the same delete by query operation you have in 'Now' would have to be broadcast to all shards to ensure there are no duplicate unique ids in the collection. It looks like that would result in the same overhead as using the composite id. Do you have any ideas on how to handle that during an update? Adding a separate shardKey definition to the schema would also cascade the change to the real-time get handler, which currently only uses the unique document ids as an input. Regarding date-based sharding, I look at that as being handled differently. With hashing a document is being assigned to a specific shard from a set of known shards where with date-based sharding I would imagine one would want to bring up a new shard for a specific time period, perhaps daily or hourly. I can imagine that it might be desirable in some cases to merge shards with older date ranges together as well if the use case favors recent updates at search time.
label: code-design
15. Updated patch (SOLR-2592_rev_2.patch) to remove custom similarities from the test configs as they were causing the test to hang.
16. Sorry for the delay on this - I have not follow it closely previously. I'll try and look closer soon - just did a quick scan tonight.
17. Looking back, I see why I missed this - I had always thought Noble intended this to be simply on the search side of things - what shards are consulted on a distrib search - which is why he said it was solved by another issue that made that pluggable. I did not realize you had started running with the pluggable hash for indexing here.
18. Maybe it is a little late to give this kind of input, and I have to admit that I did not read all comments already on this issue, but: ElasticSearch has this problem solved already separating "unique id" and "routing". It works very well, and maybe you would benefit from taking a look at how they do. There are some potentially problematic issues with this. If we allow to dynamic routing, either that the properties of the document that routing is based on is allowed to change or if the routing-rule itself is allowed to change (while you already have documents in your collection), you have a hard time making sure that updated documents are at any time living in the correct slice and that old versions of the document are not still living in other slices that used to be the place where the document had to live. Example Routing-rule: hash(doc.lastname)%number_of_slices_in_collection Events: - Document with a hash(lastname)%#slices equal to 5 is inserted into Solr. The document is stored in slice no 5. - Client loads document and changes lastname, so that hash(lastname)%#slices is now 3. From now on this particular document needs to be stored in slice no 3 (potentially running in different Solr instances than the ones running slice no 5). And you need to make sure the old version of the document is deleted on slice 5. In this case it is not simple to document-based synchronication (the bucket magic), preventing two clients from making two concurrent updates of the same document leading to inconsistency. Version-check/optimistic locking is very hard. ElasticSearch (as I remember) solves this by allowing the routing to be based on document properties, but the routing is

calculated once an for all on insertion time, and is stored as a "routing"-field on the document. If you later change the document-fields that the routing is based on the routing itself is not changed. This way documents never change where they live, but basically you cannot use the routing-rules to decide where to find your document. My advice is to make sure routing of a document cannot change. So when it is created in a slice it will live there forever, no matter what kind of updates are made on the document. If routing rules are based on property values, to keep consistency between document location and property value, you should not allow properties that routing-rule is based on to change. Regards, Per Steffensen

19. **body:** Requiring that the document never move to a different slice is certainly ideal but a very important use case involves field collapsing. The only way to make the counts work for field-collapsed results in a multi-shard setup is to ensure that docs with the same to-be-collapsed value reside on the same slice. If that field changes we may have to move the document, I don't see any way around that (unless someone can make field collapse counts work in distributed setups). Does ElasticSearch handle field-collapse for multi-shards?
    label: code-design
20. Thanks for taking a look at this Mark. I just returned from some time off and am in the process of updating the patch to the latest in branch_4x. I expect to have an update tomorrow.
21. I've updated my patch which applies cleanly to r1373086. There is a change to the AbstractZkTestCase class that allows for specifying the specific schema/solrconfig to use during the test.
22. Hi, We also have a use case where we need to divvy the index into specific slices such that a document ends up on a specific slice, in a way that all docs on that slice are related, in effect a filter of the collection of docs. Unfortunately this patch didn't meet our needs, so I've written one (SOLR_2592_solr_4_0_0_BETA_ShardPartitioner)that allows a user to create a ShardPartitioner to be more flexible about which shard a document might live. ShardPartitioner is pluggable and added via solrconfig.xml. The default is a HashShardPartitioner that works much like the existing implementation, and as an example I've also added a NamedShardPartitioner. This might be used to partition by date for example, or any other field value. It's against 4.0 Beta since that's what we're using, and meant changing some of the core testing src to test NamedShardPartitioner. Perhaps someone might be able to take a look and comment? Cheers, Dan
23. Thanks for the additional patch James. It looks like the difference in approach between your patch and the one I've submitted is that rather than hashing on a value to determine the destination shard for an update, the shard name is added as a field on a document to explicitly name the destination shard. Is that accurate? It is possible to combine our two approaches, however for explicitly naming the destination shard for an update perhaps adding optional properties to the AddUpdateCommand would be a better approach than to use a field in a document. The same could be done for a DeleteUpdateCommand to direct deletes to a specific shard(s) if desired. This would make index modification requests similar to a query where the names of the shards to be queried are present in the shards parameter as described on the wiki. The realtime get component would have to honor the shards parameter if present on a request as well, as without a way to determine shard membership from the unique id all requests to the realtime get component will have to be forwarded to all of the shards in the collection. Thoughts, anyone?
24. -1 Naming the shard inside the shard makes it impossible to split or merge shards.
25. Good point Lance, I had not even thought of that. Dan - is there a reason the patch I submitted will not work for your use case? I have a similar case where I need to ensure that docs that are related are in the same shard, and with my patch that can be done by making the document's unique id a composite id that can be parsed to extract the value that is hashed on to determine shard membership. During any subsequent split or merge operation the unique id can again be parsed (or the value of a field read) to determine membership rather than the entire unique id.
26. I've updated my patch to r1384367 on branch_4x. The only change of note is that there are no longer any changes to AddUpdateCommand as the method getHashableId has been committed as part of SOLR-3773.
27. I think I should reiterate that the default is the HashShardPartitioner, the NamedShardPartitioner was supplied as an example, and does what we needed. HashShardPartitioner partitions by hash(id) % num_shards much as the existing implementation. NamedShardPartitioner sends the doc to a particular shard, so that e.g. shard Sep2012 ONLY contains docs with doc.shard=Sep2012. Docs with doc.shard=Oct2012 would live in another shard. I think this works much the way Lance pointed out on 07/Jun/12 04:49. Michael - The problem for us with the patch you've submitted for the composite id, is that it still uses hashing to determine the shard to reside. On the indexing side, hashes of the composites might mean that e.g. doc=1234_Sep2012 and doc=4567_Oct2012 might end up in the same hash range and hence on the same shard, one might even end up with ALL docs on the same shard for example. On the searching side, again as hashing is used, it's not a simple task to determine which shard docs for Sep2012 would reside and so a query would need to be sent everywhere which would be less efficient, perhaps by a large margin, than sending the query directly to the shard. With the NamedShardPartitioner I think that since we know that all related docs live on the same shard, it should be more obvious how to split/merge shard indicies if desired. These are just two

implementations, but since we asbstract ShardPartitioner, a developer can write something that more suits their needs.

28. Dan, perhaps you could just reverse the composite id portions and use a filter query to restrict your query to a subset of the data in the shard. Using your example you would have unique ids doc=Sep2012_1234 and doc=Oct2012_4567, with each document containing a field with the values Sep2012 and Oct2012 respectively. In that way, if you started with a small amount of shards and both of those documents wound up on the same shard the search would be restricted to just the shard where they reside, and the filter would be applied to only include the docs you want. In the patch I submitted, if you only wanted to query the documents from Sep2012 you would add the parameter shard.keys=Sep2012 and the appropriate filter query. I take that exact approach with user-specific data to ensure all docs for a specific user reside in the same shard and the query is only executed against that shard and returns docs for only that user. The downside with that approach for your use case would be with a potentially low number of unique values that are hashed you could wind up with an uneven distribution of data across the shards. It sounds like what you really want is a date-based distribution policy that will add new shards to the collection each month. Does that sound about right?
29. Why should we have a composite id ? Why not we just configure the shard.key as a different field from the uniqueKey and pass the value of the field (shard.key.val) for add/delete?
30. That was my initial thought as well, however if the value that is hashed on to determine shard membership cannot be extracted from the unique id, the realtime get handler would have to broadcast the query to all shards in the collection. Perhaps the shard.keys parameter could be added to the realtime get handler, but that seems to be counter to the handler's purpose.
31. Realtime get should not be an overriding factor for designing this feature. Let it be as follows . if shard.key.val is passed along for realtime get use that to identify the membership else do a distributed search .
32. Thanks for the feedback Noble! I will update the patch I submitted with the following: * Add the definition of a shardKey field in the schema, similar to how the unique id is defined. If no shardKey field is defined in the schema, the value hashed will fall back to the unique id. * Add support for an optional shard.key.val request parameter to the realtime get handler, if the value is not specified the query will be distributed. * Add support for an optional property on the delete command for shard.key.val, if the value is not specified the delete will be distributed. * Remove the pluggable implementation on how the value to be hashed is determined.
33. **body:** This issue should be about "custom hashing". "Custom sharding" is different, and covers many other things such as time-based sharding. I haven't had a chance to look at the patches here, but it seems like simpler may be better. I like the approach of just hashing on the id field. For example, if you have an email search application and want to co-locate all a users data on the same shard, then simply use id's of the form userid:emailid (or whatever separator we choose - we should chose one by default that is less likely to accidentally clash with normal ids, and also one that works well in URLs and hopefully in query parser syntax w/o the need for escaping). And then when you hash, you simply use the upper bits of the userid and the lower bits of the emailid to construct the hash that selects the node placement. The only real configurable part you need is where the split is (i.e. how many bits for each side).
    label: code-design
34. yonik. I feel we are dictating the schema of users with this suggestion. isn't it straight forward when we say shard on an arbitrary field but just pass the value of that field if you want optimized look up
35. bq. isn't it straight forward when we say shard on an arbitrary field No, I think that makes things very difficult. Every place where you used to have "id" you now need to pass "id" and this other configurable field (or maybe even more than one). That puts much more of a burden on clients trying to talk to solr since "id" is no longer good enough to efficiently identify where a document lives in the cluster.
36. **body:** Think about an app whose schema and key are already defined and they are moving to cloud, this will force them to change their keys. Then we have a completely arbitrary way of how to parse the key depending on how it is encoded. For the clients , it is equally easy or hard to pass an extra param than constructing a key that contains an extra parameter. Most likely the key is the only thing that they use in the app (say front end) . I think, it is ugly and inconvenient to dictate how the key should look like bq.(or maybe even more than one) We can enforce that only one field can be used as a "shard.key"
    label: code-design
37. It is not just realtime get that we need to support. What if I need to fire a search for a specific shard (say data for a single user). Searches limited to a single shard is a more common usecase . The query will not have the uniqueKey parameter. In that case we will need to pass the shard.key.val as a separate param. So to keep it uniform , let us not mangle the uniqueKey
38. The idea of a shard.key is what I did with the supplied patch, e.g. <shardPartitioner name="ShardPartitioner" class="org.apache.solr.cloud.NamedShardPartitioner"> <str name="shardField">date</str> </shardPartitioner> Though we could use any field, region,date etc. It's NOT specifically about date partitioning and it's at the users discretion. The default is a HashPartition: hash(id) % num_shards Michael -

Your suggestion on 15/Sep/12 02:36 for us still wouldn't address the issue of knowing exactly on what shard a doc lives. For our (and I guess for most) apps, most queries are search ones, and we'd need to send a query to every shard, but in our app, I already know in advance what subset of the index I need to search, and to speed the query up I'd want to index docs that way too so that I ONLY need to query a particular shard. If I know the subset in advance, anything with fq=... seems wasteful to me. The downside of my implementation is that deletes and RealTimeGets would be slower since the id alone is not enough to determine shard membership, and hence needs to be sent everywhere, but I suspect in most applications, this is a welcomed compromise as most queries will be search ones. Perhaps shard membership can be efficiently stored in a distributed bloom filter or something like, to speed that up? All this aside, as a compromise I've though that for us we can take this one level higher, i.e. instead of collections=docs and shard=Aug2012,Sep2012 etc we can do collections=docs_Aug2012,docs_Sep2012. Then if we need to search across multiple dates, we can do this today, and still have hashed based sharding, by using collection=docs_Aug2012,docs_Sep2012,... in the query. Others might find this idea useful too.

39. Michael, I've been using your patch for a few months now (since it's the only way we can get field collapse counts to work properly for distributed queries) and it looks like your latest thoughts are similar to the discussions earlier (~ June 23)? If I understand correctly, all we'll need to do is add the shardKey to our schema and we no longer will need the composite uniqueKey and corresponding ShardKeyParsers? We'll still need to deal with the situation where our document's shardKey field changes (to relocate the doc to a different shard, etc.) but I see no way around that. Regarding realtime get, are you aware of [SOLR-3133|https://issues.apache.org/jira/browse/SOLR-3133]? Just wondering if the use of the shard.key.value parameter will help or hinder that issue (there's not a lot of detail about why RT get doesn't work in distributed mode and I haven't had a chance to look myself).
40. The latest patch I submitted (SOLR-2592_r1384367.patch) still requires encoding the value to be hashed within the unique id such as with a composite id. I am more in the mindset of keeping it simple by encoding the hashed value into the document's unique id, and will defer to the committers provide guidance in the approach taken. At the end of the day, either solution will address my need of ensuring related documents are all on the same shard and I can query a given subset of shards rather than the entire collection. I was not aware of SOLR-3133, however it looks to be a duplicate of SOLR-2656 which was committed on March 27, 2012, giving the realtime get handler distributed support.
41. Just want to recap thoughts on all the different types/levels of custom sharding/hashing: 1) custom sharding with complete user control... user is responsible for adding/removing shards, and specifying what shard an update is targeted toward (SOLR-4059) - example: http://.../update?shard=NY_NJ_area - Solr still keeps track of leader, still forward updates to the leader which forwards to replicas - replicas can still be added on the fly - Users could also provide a pre-built shard and we could still replicate it out - search side of this is already implemented: ?shards=NY_NJ_area,SF_area - OPTIONAL: we could still provide a shard splitting service for custom shards - OPTIONAL: somehow specify a syntax for including the shard with the document to support bulk loading, multiple docs per request (perhaps magic field _shard_?) - if we go with _shard_, perhaps we should change the request param name to match (like we do with _version_?). Example: http://.../update?_shard_=NY_NJ_area 2) custom sharding based on plugin: a superset of #1 - a plugin looks at the document being indexed and somehow determines what shard the document belongs to (including possibly consulting an external system) - an implementation that takes the shard name from a given field - IDEA: plugin should have access to request parameters to make decision based on that also (i.e. this may be how _shard_ is implemented above?) - atomic updates, realtime-get, deletes, etc. would need to specify enough info to determine shard (and not change info that determines shard) - trickier for parameters that already specify a comma separated list of ids... how do we specify the additional info to determine shard? - OPTIONAL: allow some mechanism for the plugin to indicate that the location of a document has changed (i.e. a delete should be issued to the old shard?) - is there a standard mechanism to provide enough information to determine shard (for example on a delete)? It would seem this is dependent on the plugin specifics and thus all clients must know the details. 3) Time-based sharding. A user could do time based sharding based on #1 or #2, or we could provide more specific support (perhaps this is just a specific implementation of #2). - automatically route to the correct shard by time field - on search side, allow a time range to be specified and all shards covering part of the range will be selected - OPTIONAL: automatically add a filter to restrict to exactly the given time range (as opposed to just the shards) - OPTIONAL: allow automatically reducing the replication level of older shards (and down to 0 would mean complete removal - they have been aged out) 4) Custom hashing based on plugin - The plugin determines the hash code, the hash code determines the shard (based on the hash range stored in the shard descriptor today) - This is very much like option #2, but the output is hash instead of shard 5) Hash based on field (a specific implementation of #4?) - collection defines field to hash on - OPTIONAL: could define multiple fields in comma separated list. the hash value would be constructed by catenation of the values. - how to specify the restriction/range on the query side? 6) Hash based on first part of ID (composite id) - Like #5, but the hask key value is contained in the ID. - very transparent and unobtrusive - can be enabled by default since there should be no additional

requirements or restrictions For both custom hashing options #5 and #6, instead of deriving the complete hash value from the hash key, we could use it for only the top bits of the hash value with the remainder coming from the ID. The advantage of this is that it allows for splitting of over-allocated groups. If the hash code is derived only from the hashKey then all of a certain customer's records would share the exact same hash value and there would be no way to split it later. I think eventually, we want *all* of these options. It still seems natural to go with #6 first since it's the only one that can actually be enabled by default w/o any configuration. Other things to think about: Where should the hashing/sharding specification/configuration be kept? a) as a collection property (like configName)? b) as part of the standard "config" referenced by configName (either part of the schema or a separate file more amenable to live updating) Handing grouping of documents by more than one dimension: Let's say you have multiple customers (and you want to group each customers documents together), but you also want to do time based sharding.

42. Hi Michael, while I was reviewing your patch, I realized that we want something that clients (like cloud aware SolrJ) can easily get to. So instead of config in solrconfig.xml that defines a parser for the core, it seems like this should be a property of the collection? The HashPartitioner object is currently on the ClusterState object, but it really seems like this needs to be per-collection (or per config-set?). Currently, there is a /collections/collection1 node with {code} {"configName":"myconf"} {code} We could add a "partitioner" or "shardPartitioner" attribute. And of course there is /clusterstate.json with {code} {"collection1":{ "shard1":{ "range":"80000000-ffffffff", "replicas":{"192.168.1.109:8983_solr_collection1":{ "shard":"shard1", "roles":null, "state":"active", "core":"collection1", "collection":"collection1", "node_name":"192.168.1.109:8983_solr", "base_url":"http://192.168.1.109:8983/solr", "leader":"true"}}}, "shard2":{ "range":"0-7fffffff", "replicas":{}}}} {code} Now, currently the ClusterState object is created by reading clusterstate.json, but I don't believe it reads /collections/<collection_name>, but perhaps we should change this? The other issue is that there is currently no java Collection object (it's just a map of the contained shards) to expose info like what the hash partitioner used is. I think I'll start by trying to refactor some of the ClusterState code to introduce that.
43. **body:** bq. Now, currently the ClusterState object is created by reading clusterstate.json, but I don't believe it reads /collections/<collection_name>, but perhaps we should change this? Depending on the number of collections, it may add a lot of reads on cluster changes, but the Overseer could probably grab this from the collection node and add it to the cluster state json file? It could perhaps even use a watcher per collection and not read when it does not have to. Or it could be read separately like live nodes - that gets a bit messy though.
label: code-design
44. **body:** bq. the Overseer could probably grab this from the collection node and add it to the cluster state json file? Yeah, that's the other way, and might be initially easier. To keep back compat, we would need to add a magic name to the map that we know isn't a shard. "properties"?
label: code-design
45. bq. I realized that we want something that clients (like cloud aware SolrJ) can easily get to. So instead of config in solrconfig.xml that defines a parser for the core, it seems like this should be a property of the collection? Agreed - that makes perfect sense and was an oversight on my part. Persisting that information in /collections/<collection_name> makes sense as it is a collection level option. One thing about my patch that has been on my mind has been the parsing of the value to hash on from a delete by query on the client side. Currently it is very simple and uses a regex to extract the value as using any of the query parsers would require the cloud aware SolrJ client to also have the Lucene jar, which I'm not sure is desired.
46. FWIW, I'm trying out github for this patch. My current changes to add a DocCollection abstraction to ClusterState are on the "custom_hash" branch in my lucene-solr fork: https://github.com/yonik/lucene-solr/tree/custom_hash https://github.com/yonik/lucene-solr/commit/5f82a7917862a1f9e70d6d268c44b23af18aca3b Warning: it doesn't work yet... I just got it to compile, but tests are failing.
47. OK, fixed the bugs: https://github.com/yonik/lucene-solr/commit/3c0de9db3f737ee24a8f47ab3db9c573a173ce7d branch is back to working.
48. Latest issue I think I'm running into is manual shard assignment... when a core comes up and says "hey, I'm foo-shard of bar-collection", and that collection is dynamically created on the fly since it doesn't exist yet. I think we have to assume "custom sharding" in that case... assigning hash ranges doesn't make sense. But our current indexing code happily splits up all of the shards at the time (without recording any hash range because numShards was never passed) and indexes to them based on that (which is a no-no if new shards can be manually added at any time). I guess in situations like this, we should create the collection with an "implicit" document router. The shard the document belongs to is defined by the shard it's sent to (it's custom sharding with the shard coming from the URL essentially). Before I diverge too much from trunk, I think I may try to first clean up what I have and get that committed back (the introduction of collection properties and associated cleanups).
49. Progress patch attached. - DocCollection object introduced instead of generic ZkProps - "properties" added for collections - a "router" collection property - HashPartitioner renamed DocRouter, and hopefully

represents the single plugin to implement for all types of custom hashing/sharding - some built in routers such as compositeIdRouter All tests currently pass (since the router on the collection isn't yet being used). I plan to commit soon so there won't be such a big merging mess later.

50. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1416025 SOLR-2592: progress - introduce DocCollection, add properties for collections, add a router collection property, add some builtin routers, rename HashPartitioner to DocRouter
51. Hmmm, I've committed to trunk, but having issues merging back to 4x. I got a conflict in AbstractFullDistribZkTestBase.java (and 4x does look like it varies a lot from trunk). Getting test failures in things like BasicDistributedZkTest even after resolving conflicts, so this could be a deeper issue of too much divergence between 4x and trunk.
52. I just noticed that with these changes, the cloud viz ui needs to be updated - it sees properties as a shard name and displays it along with the actual shards.
53. If I remember right, when I got the directory with replication stuff working and more tests using ram dir and what not, I started seeing unrelated test failures pop up and has to do some test work to address them. I probably could have back ported all of those, but hadn't yet. I was hoping to merge all that stuff back by now but have been waiting for it to bake a bit.
54. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1416216 SOLR-2592: refactor doc routers, use implicit router when implicity creating collection, use collection router to find correct shard when indexing
55. Here's the work I've done handle the query side (while referencing Michael's patch for changes to ShardHandler). Which slices to query should now be delegated to the router for the collection. Something is off though - BasicDistributedZkTest fails (although not all of the time!). This patch is to just let people know the direction I'm going to give early feedback for things like the DocRouter interface.
56. bq. Something is off though False alarm - there was some sort of non-reproducible build issue. I've committed the latest patch.
57. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1416744 SOLR-2592: delegate query-time shard selection to collection router
58. Hi All, I am trying to use the patches mentioned here. Specifically the one by Michael which hashes on a combined field. (https://issues.apache.org/jira/browse/SOLR-2592?focusedCommentId=13280334&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-13280334) My requirement is exactly as mentioned by Andy https://issues.apache.org/jira/browse/SOLR-2592?focusedCommentId=13294837&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-13294837 I have downloaded the latest Nightly build but am not sure how to configure it. Are the patches on the nightly build? (https://builds.apache.org/job/Solr-Artifacts-4.x/lastSuccessfulBuild/artifact/solr/package/) How and where in solrconfig do I configure <shardKeyParserFactory class="solr.ShardKeyParserFactory"/> ?
59. Hi Shreejay - I have not updated my patch in a few months so it may not apply cleanly to the latest in branch_4x. The last patch I updated is SOLR-2592_r1384367.patch. If it does apply cleanly you can configure the shard key parser as a child of the 'config' element. Here is an example that will work with a composite unique id delimited with underscores such as "123_abc", and the document will be hashed based on "123". The clause section of the config will be used in a delete by query, and in the example below the value of clause for the field "foo_field" will be used to hash on to determine which shard the delete will be forwarded to. If the clause is required and a delete by query is submitted by a client the request will return an error. {code:xml} <shardKeyParserFactory class="org.apache.solr.common.cloud.CompositeIdShardKeyParser"> <str name="clause">foo_field</str> <bool name="clauseRequired">false</bool> </shardKeyParserFactory> {code} That being said the implementation is going in a different direction that will move the custom hashing configuration to be persisted within Zookeeper to allow a client to be aware the custom hashing on the collection without having to parse the solrconfig. I have not yet had the chance to review the work Yonik has committed so far.
60. Thanks for the quick reply. For some reason it does not seem to be working on my local machine. In the above example is foo_field the composite field? or the field on which the hash is done? But like u mentioned the "clause" field only matters during deletes. I am trying to add documents and they are not going to same shards. I have created a composite unique key (unique_id = guidid + "_" + anotherID) guidid is a string guid. My schema file has these two lines. <field name="unique_id" type="string" indexed="true" stored="true" required="true" /> <uniqueKey>unique_id</uniqueKey> In my Solrconfig file under <config> i have added the below lines. <shardKeyParserFactory class="org.apache.solr.common.cloud.CompositeIdShardKeyParser"> <str name="clause">guidid</str> <bool name="clauseRequired">false</bool> </shardKeyParserFactory> Am I missing configuring anything else? I have applied the SOLR-2592_r1384367.patch on branch_4x and the build was successful.

61. Shreejay, in your example the value that would be hashed to determine shard membership would be the "guidid", and those that hashed to the same value would be on the same shard. The foo_field is a different field from the unique id and its value would be the same as that is hashed. For example, in a system that will keep all documents that belong to a specific user id on the same shard, the unique, composite id would be <userId>_<otherIdData> and the clause field would contain the user id to allow for a delete by query using the user id as a clause in the query to forward the delete to only the shards that contain that data. Did all of the tests pass on your build? There are tests in my patch that will verify it works properly.
62. Thanks Michael. So my configuration seems to be good. I just downloaded the source and applied the patch again. I am running the tests again now. Last time I ran them it got stuck at these messages for a long time. So I cancelled it. I will let it run this time till the finish. only … [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:32:15, stalled for 2508s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:33:16, stalled for 2569s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:34:16, stalled for 2629s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:35:16, stalled for 2689s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:36:16, stalled for 2749s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:37:16, stalled for 2809s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads [junit4:junit4] HEARTBEAT J0: 2012-12-06T13:38:16, stalled for 2869s at: TestInd exWriterNRTIsCurrent.testIsCurrentWithThreads
63. Hi Michael, I ran the tests for solr after compiling it with the patch. A few of the tests failed. I am trying to fix these. If anyone else has used this patch successfully without errors, please let me know. [junit4:junit4] Completed in 0.10s, 1 test, 1 skipped [junit4:junit4] [junit4:junit4] [junit4:junit4] Tests with failures: [junit4:junit4] - org.apache.solr.handler.admin.ShowFileRequestHandlerTest.tes tGetRawFile [junit4:junit4] - org.apache.solr.handler.admin.ShowFileRequestHandlerTest.tes tDirList [junit4:junit4] - org.apache.solr.cloud.ZkCLITest.testBootstrap [junit4:junit4] - org.apache.solr.cloud.BasicDistributedZkTest.testDistribSear ch [junit4:junit4] - org.apache.solr.cloud.BasicDistributedZkTest (suite) [junit4:junit4] [junit4:junit4] [junit4:junit4] JVM J0: 1.87 .. 2645.62 = 2643.75s [junit4:junit4] Execution time total: 44 minutes 10 seconds [junit4:junit4] Tests summary: 238 suites, 983 tests, 3 suite-level errors, 3 er rors, 1 failure, 426 ignored (4 assumptions) BUILD FAILED
64. **body:** bq. That being said the implementation is going in a different direction that will move the custom hashing configuration to be persisted within Zookeeper to allow a client to be aware the custom hashing on the collection without having to parse the solrconfig. I have not yet had the chance to review the work Yonik has committed so far. Right - I've made good progress on that front in trunk, and we'll figure out how to get it ported back to 4x. I'm in the process of adding more tests right now. The separator: I went with "!" by default since it doesn't require URL encoding, but is less common than underscore. It also reminded me of the bang paths of old-style UUCP email addresses (like bigco!user). A composite id router is enabled by default since the bangs are optional. For querying, I used the param that Michael started with, "shard.keys". So you can do shard.keys=bigco!,littleco! at query time. (the bangs matter! leaving it out will simply query the shard containing a simple document id, while putting it in will query a range of documents all covered by that domain). I've added distributed short-circuiting as well... if you only need to query a single shard, and you send the query directly to a replica of that shard that is active, the distributed search phase will be skipped and we'll drop directly to a local search.
    label: code-design
65. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418507 SOLR-2592: more DocCollection refactoring in tests
66. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418337 SOLR-2592: fix composite id router for full-range
67. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418116 SOLR-2592: fix composite id router slice selection
68. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418100 SOLR-2592: fix shifting by 32 bits in compositeId router
69. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418093 SOLR-2592: add hash router tests, fix compositeId bit specification parsing
70. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418043 SOLR-2592: refactor routers to separate public classes
71. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418030 SOLR-2592: avoid distrib search if current core is active and hosting targeted shard
72. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418755 SOLR-2592: integration tests for routing

73. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418762 SOLR-2592: realtime-get support
74. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1418814 SOLR-2592: deleteByQuery routing support
75. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1419019 SOLR-2592: fix implicit router to return found slice, test querying via shards
76. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1420284 SOLR-2592: cleanups - remove unneeded range info, rename getLeaderProps to getLeaderRetry
77. Hi Yonik, What configuration (solrconfig / solr.xml /schema.xml / ZK ?) do I have to change in order to use these changes in trunk? I have a string field on which I want to do "group by" and hence I want all documents with same value of the string field to be on same side. I was using a 4x branch with Michael's patch, but ran into heap space issues mentioned in https://issues.apache.org/jira/browse/SOLR-4144 . I was not able to merge trunk to 4x and just saw your comment that you also had issues while doing this. So my next step might be to use the trunk directly with the patch supplied for 4144. How do I specify that I want to shard based on a specific string field? Thanks. --Shreejay
78. **body:** Great to see someone that can help with additional testing :) This will get merged back to 4x very soon for 4.1. It's a pain because of some work I did around full support for custom Directories. Merging both back at the same time will probably be easiest.
    label: code-design
79. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1420338 SOLR-2592: avoid splitting composite router hash domains
80. So with the composite key hashing strategy of using the first part of the ID as the upper bits (what I've started calling a hash domain) and the second part the lower bits, one needed to use an initial numShards equal to a power of two *if* one wanted to absolutely avoid a hash domain from being split over more than one shard. For most people, this wouldn't be an issue - you only need to guarantee splitting domains if you're using a feature that doesn't work across shards (like pseudo-join for example). I just checked in a change to remove that limitation. You can now use any numShards and the resulting shard ranges will be rounded to the nearest hash domain to avoid splitting them (this amounts to a maximum size rounding error of 1/65536th of the hash ring, or 0.002%).
81. I use the hot shard concept in Solr 3.5.0. For the cold shards, I split documents using a MOD on the CRC32 hash of a MySQL bigint autoincrement field - my MySQL query does the CRC32 and the MOD. That field's actual value is translated to a tlong field in the schema. For the hot shard, I simply use a split point on the actual value of that field. Everything less than or equal to the split point goes to the cold shards, everything greater than the split point goes to the hot shard. Multiple shards are handled by a single Solr instance - seven shards live on two servers. This arrangement requires that I do a daily "distribute" process where I index (from MySQL) data between the old split point and the new split point to the cold shards, then delete that data from the hot shard. Full reindexes are done with the dataimport handler and controlled by SolrJ, everything else (including the distribute) is done directly with SolrJ. How much of that could be automated and put server-side with the features added by this issue? If I have to track shard and core names myself in order to do the distribute, then I will have to decide whether the other automation I would gain is worth switching to SolrCloud. If I could avoid the client-side distribute indexing and have Solr shuffle the data around itself, that would be awesome, but I'm not sure that's possible, and it may be somewhat complicated by the fact that I have a number of unstored fields that I search on. At some point I will test performance on an index where I do not have a hot shard, where the data is simply hashed between several large shards. This entire concept was implemented for fast indexing of new data - because Solr 1.4 did not have NRT features.
82. [branch_4x commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1420992 SOLR-2592: SOLR-1028: SOLR-3922: SOLR-3911: sync trunk with 4x
83. **body:** Here's a simple patch that implements adding an extra level for each collection, as discussed here: http://find.searchhub.org/document/2e6ee3810c83aeeb
    label: code-design
84. Here's an update to the properties patch that handles the GUI change (cloud.js)
85. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1421499 SOLR-2592: add additional level in clusterstate.json for collection properties
86. [branch_4x commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1421506 SOLR-2592: add additional level in clusterstate.json for collection properties
87. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1421644 SOLR-2592: test expected number of requests to servers
88. [branch_4x commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1421670 SOLR-2592: test expected number of requests to servers
89. I think this belong to this issue. We're seeing with today's trunk and a clean Zookeeper the following exception. For some reason, only one node exhibits this behavior. {code} 2012-12-14 10:10:27,355 ERROR

[apache.zookeeper.ClientCnxn] - [main-EventThread] - : Error while calling watcher java.lang.ClassCastException: java.lang.String cannot be cast to java.util.Map at org.apache.solr.common.cloud.ClusterState.makeSlices(ClusterState.java:246) at org.apache.solr.common.cloud.ClusterState.collectionFromObjects(ClusterState.java:231) at org.apache.solr.common.cloud.ClusterState.load(ClusterState.java:219) at org.apache.solr.common.cloud.ZkStateReader$2.process(ZkStateReader.java:207) at org.apache.zookeeper.ClientCnxn$EventThread.processEvent(ClientCnxn.java:519) at org.apache.zookeeper.ClientCnxn$EventThread.run(ClientCnxn.java:495) {code} Restarting the entire cluster and cleaning ZK does not help. This single node remains stuborn and cannot get up. It also simply stops logging anything after including the regular ping requests.

90. Ah! For some reason this specific node did not get the upgrade from December 13th to the 14th while all others did. This proves the change is indeed backward incompatible ;) The problem is solved so please ignore my previous comment.
91. Additional note for anyone: you will also see a similar exception with an older SolrJ application: {code} java.lang.ClassCastException: java.lang.String cannot be cast to java.util.Map at org.apache.solr.common.cloud.ClusterState.load(ClusterState.java:291) at org.apache.solr.common.cloud.ClusterState.load(ClusterState.java:263) at org.apache.solr.common.cloud.ZkStateReader.createClusterStateWatchersAndUpdate(ZkStateReader.java:274) at org.apache.solr.client.solrj.impl.CloudSolrServer.connect(CloudSolrServer.java:142) at org.apache.nutch.indexer.solr.SolrUtils.getCloudServer(SolrUtils.java:107) at org.apache.nutch.indexer.solr.SolrUtils.getSolrServers(SolrUtils.java:65) at org.apache.nutch.indexer.solr.SolrWriter.open(SolrWriter.java:65) at org.apache.nutch.indexer.IndexerOutputFormat.getRecordWriter(IndexerOutputFormat.java:42) at org.apache.hadoop.mapred.ReduceTask$OldTrackingRecordWriter.<init>(ReduceTask.java:448) at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:490) at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:420) at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:260) {code}
92. **body:** {quote}Right - I've made good progress on that front in trunk, and we'll figure out how to get it ported back to 4x. I'm in the process of adding more tests right now. The separator: I went with "!" by default since it doesn't require URL encoding, but is less common than underscore. It also reminded me of the bang paths of old-style UUCP email addresses (like bigco!user).{quote} Can the separator be provided as an option instead of being hard coded? I have been using Michaels patch and now I am trying to move to 4.1. But I will have to re-index all my data by changing the unique key, which right now contains a underscore. If not, will changing the separator to a underscore in CompositeIdRouter.java be enough? --Shreejay
    label: code-design
93. [trunk commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1433082 SOLR-2592: changes entry for doc routing
94. [branch_4x commit] Yonik Seeley http://svn.apache.org/viewvc?view=revision&revision=1433084 SOLR-2592: changes entry for doc routing
95. [trunk commit] Steven Rowe http://svn.apache.org/viewvc?view=revision&revision=1434401 - Make complex SOLR-2592 changes entry not get converted to a single wrapped line in Changes.html. - 'Via' -> 'via' (merged lucene_solr_4_1 r1434389)
96. [branch_4x commit] Steven Rowe http://svn.apache.org/viewvc?view=revision&revision=1434402 - Make complex SOLR-2592 changes entry not get converted to a single wrapped line in Changes.html. - 'Via' -> 'via' (merged lucene_solr_4_1 r1434389)
97. I don't see this changesin the code. I use solr 4.3.1, but I looked for this changes in solr 4.1 too.