Item 248
git_comments:

git_commits:

1. **summary:** Add hyper app to dependencies
   message: Add hyper app to dependencies Releases and dialyzer checks need app dependencies to work properly Issue: #1346

github_issues:

github_issues_comments:

github_pulls:

1. **title:** Add _approx_count_distinct as a builtin reduce function
   body: ## Overview We've seen a number of applications now where a user needs to count the number of unique keys in a view. Currently the recommended approach is to add a trivial reduce function and then count the number of rows in a _list function or client-side application code, but of course that doesn't scale nicely. It seems that in a majority of these cases all that's required is an approximation of the number of distinct entries, which brings us into the space of hash sets, linear probabilistic counters, and the ever-popular "HyperLogLog" algorithm. Taking HLL specifically, this seems like quite a nice candidate for a builtin reduce. The size of the data structure is independent of the number of input elements and individual HLL filters can be unioned together. There's already what seems to be a good MIT-licensed implementation on GitHub: https://github.com/GameAnalytics/hyper This PR adds a new `_approx_count_distinct` function which uses the binary HLL implementation above. It has the precision fixed to 11 which yields a relative error of 2%. A future improvement would make this precision configurable in the design document. ## Testing recommendations 1. Generate a view with a known number of distinct keys. 2. Add `_approx_count_distinct` as a reduce function. 3. Compare the reduce output with the exact number of distinct keys. The result should be within 2%. ## Related Issues or Pull Requests See https://issues.apache.org/jira/browse/COUCHDB-2971 for history. ## Checklist - [x] Code is written and works correctly; - [x] Changes are covered by tests; - [x] Documentation reflects the changes (separate PR forthcoming).

github_pulls_comments:

1. Noticed a transitive dependency on bisect: ``` (HEAD detached at CouchDB-2.2.0)) $ more rebar.config {cover_enabled, true}. {deps, [{bisect, "", {git, "https://github.com/knutin/bisect.git", {branch, "master"}}}]}. ``` Which brings in proper master dependency, which fails to compile on R16. However the good news is we don't use the bisect backend just the default (binary one). So I think we can make a branch on our version of hyper to exclude bisect and it should be good. Another one might be to avoid building carray C module (since we don't use it) but that's optional.
2. Good catch @nickva I totally missed that dependency. Have removed it now. Left the carray to minimize the fork from upstream.
3. Added some tests for this feature: https://github.com/apache/couchdb/pull/1364
4. A small PR to add hyper to top level `.gitignore` file: https://github.com/apache/couchdb/pull/1365
5. **body:** So I realized we have one issue we need to discuss here: collation. The reduce function is inserting each key into the filter using a regular old `term_to_binary(Key)`, so it will (incorrectly) treat two keys that are not identical but compare equal in ICU as distinct keys in its computation. I see a handful of paths that we could take: 1. Force the user to set `"collation": "raw"` in the design document options in order to use this reducer. 2. Merge this feature as-is and hide behind the "approximate" nature of the computation, documenting that it may overestimate the number of distinct keys by more than the stated uncertainty in the presence of such keys. 3. Compute a "canonical form" for each key and insert that into the filter instead. I don't even know if that's possible. My initial preference is for Option 2 but I wanted to bring this up before merging to see if others have strong opinions.
   label: code-design
6. I am thinking of Option 2 with the documentation note. For Options 3, Erlang 20.0 has a bunch of new normalization functions like say: http://erlang.org/doc/man/unicode.html#characters_to_nfc_binary-1. Maybe recursively transform each key to a normalized format before calling term_to_binary on it? Would that be enough? Even if it is would need to backport the module to Erlang versions < 20.0
7. **body:** This would be enough of a reason to start requiring Erlang 20 for me, in say 2.3.0 or 3.0. It could be transparent to the user; if the function is available, use it and improve the accuracy of the response.
   label: code-design
8. Very cool, I had forgotten about all of that core unicode work. Definitely a nice future enhancement.
9. FYI this broke the Windows build: ``` Compiling c:/relax/couchdb/src/hyper/c_src/hyper_carray.c 'cc' is not recognized as an internal or external command, operable program or batch file. ERROR: compile failed while processing c:/relax/couchdb/src/hyper: rebar_abort make: *** [couch] Error 1 ``` We really need to get Windows into the CI build farm somehow.
10. Since I faced the same issue, I started looking to fix that. Could it be fixed with a simple alias such as: export cc=g++ ?
11. no, it's worse than that sadly, there is some invalid pointer arith and some C99 reserved words the MS compiler doesn't support. fixing rebar.config is pretty easy, there's no need to alias CC=cc, just leave it out. I have this worked out already I'll get to it tomorrow.

github_pulls_reviews:

1. Now that we are fully read/write on GitHub is it possible to fork this repo into Apache? Or is the requirement still to clone and push a copy (and eliminate the edge in the network graph in the process).
2. We are still required to go through code clearance and clone and push a copy. We recently did this for bcrypt support, check the history.
3. I'll note the infra piece of this is now self-service assuming we vet the licensing and ensure we don't break our own buildchain. See https://gitbox.apache.org/ for the link.
4. I don't see any formal code clearance entry for bcrypt at http://incubator.apache.org/ip-clearance/, just @janl's commit to amend our overall `LICENSE` and `NOTICE` in 817b2b6. Guessing I can do the same here. The `hyper` code is fully MIT-licensed; there was an LGPL file in the C implementation but this was expunged in GameAnalytics/hyper#16
5. Done.
6. Now change `rebar.config.script` to point at our copy :) You also shouldn't point at a specific hash, but rather a tagged release. Standard practice btw is to create an `upstream` branch and tag any upstream releases on that as `X.Y.Z`, saving the `master` branch for any local changes we might have to make. If we need to make a release with our customisations, use tags of the form `COUCHDB-X.Y.Z` instead.
7. @wohali I already did all that in 22e8f97 ;) Your description of the standard practice sounds like a good one to me. In this case upstream doesn't actually have any tags (🤷‍♂️), so I picked `X.Y.Z` optimistically as `2.2.0` 😄

jira_issues:

1. **summary:** Provide cardinality estimate (COUNT DISTINCT) as builtin reducer
   description: We've seen a number of applications now where a user needs to count the number of unique keys in a view. Currently the recommended approach is to add a trivial reduce function and then count the number of rows in a _list function or client-side application code, but

of course that doesn't scale nicely. It seems that in a majority of these cases all that's required is an approximation of the number of distinct entries, which brings us into the space of hash sets, linear probabilistic counters, and the ever-popular "HyperLogLog" algorithm. Taking HLL specifically, this seems like quite a nice candidate for a builtin reduce. The size of the data structure is independent of the number of input elements and individual HLL filters can be unioned together. There's already what seems to be a good MIT-licensed implementation on GitHub: https://github.com/GameAnalytics/hyper One caveat is that this reducer would not work for group_level reductions; it'd only give the correct result for the exact key. I don't think that should preclude us from evaluating it.

jira_issues_comments:

1. That would be neat to have. So if HLLs can be unioned then they satisfy the associativity and commutativity bit. So the algorithm could be if rereduce is false, then create an HLL object as an accumulator and add all keys to it, return that. If it is true, then merge (union) together all HLL registers to produce a single HLL register (the hyper implementation above has that function). It seems they'd need to be a final transformation before returning the final result -- extract estimated cardinality from the HLL object. Multiple precisions could be hacked together as a selection of <<"_distinct_hll_4">> or <<"_distinct_hll_6"">>, etc variants.
2. **body:** Exactly! I took a shot at implementing this yesterday evening and got the basics working. The crux of the implementation is exactly as you describe: {code} count_distinct_keys(reduce, KVs) -> lists:foldl(fun([[Key, _Id], _Value], Filter) -> hyper:insert(term_to_binary(Key), Filter) end, hyper:new(Precision), KVs); count_distinct_keys(rereduce, Reds) -> hyper:union([Filter || [_, Filter] <- Reds]). {code} You're right that we need a "finalize" operation to estimate the cardinality given the final HLL object. The couch_query_servers API doesn't make this easy - the `rereduce` function is used internally by the btree reducer as well as via the public `fold_reduce` function. I ended up adding a `finalize/1` function and calling it in the two places that mattered. I'll try to post my WIP in the next 24 hours. The unions are a fairly expensive operation and so we need to be extra careful to keep the tree wide and shallow. There's an interesting interplay here between the precision of the filter and the chunking threshold for the tree; this is one case where the default chunkify function is almost certainly not optimal. It'll be more code, but I thought it might be nicer to allow the precision of the filter to be specified as an option in the view specification in the design doc instead of creating 12 "different" builtins.
   label: code-design
3. **body:** Ah, good point on having a nicer way to specify precision. Yeah otherwise it looks kind of hackish. Noticed they provide various backends for the registers. One is a C NIF. Tried to compile and run their code on Erlang 18 and had to fiddle with it a bit, but got it to work and got these results: https://gist.github.com/nickva/bf19a2b7b537f5051a99 There are some tradeoffs between memory usage, cardinality and union times. While C array is interesting, having the cheapest union operation (under 1ms), has cardinality estimation time greater than a few milliseconds which might not play well with the Erlang schedulers. But if it happens only during the finalize stage it could be handled in another way (some thread + queue mechanism). Unfortunately it also has a large/constant memory usage for low cardinalities.
   label: code-design
4. Ah, I hadn't even noticed the NIF option. Looks like it was added after the initial release by a third party, which probably explains why it's undocumented. Perhaps worth reaching out to the maintainer to see if they recommend it. I haven't kept pace with all of the long-running NIF issues as Erlang releases have evolved, but I'd hope that an infrequent invocation of the cardinality estimation function running for < 5 ms wouldn't be an issue.
5. Commit 50725f2df7d53d82ed269935aba362063940e307 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://git-wip-us.apache.org/repos/asf?p=couchdb.git;h=50725f2] Add hyper to the build COUCHDB-2971
6. we'll need an asf repo for this before merging.
7. I noticed we had ASF repos for everything. What's the thinking there?
8. We have to build releases from code hosted at the ASF (we can't allow a third party to alter the contents of our releases, which they'd be able to do if we pulled from a github account we don't own).
9. Huh, I always figured that we'd be voting on the aggregate source code obtained after `rebar get-deps update-deps`. Forking a repo and then pushing it to GitHub without acknowledging the provenance is poor form, don't you think?
10. Hm, I don't, but perhaps we should have a thread on dev@. To date, we've made ASF repo forks of all the things we need to build couchdb, whether third-party or not.
11. Commit f7c3c24db17db5908894447e1822936212d61fcd in couchdb-couch-mrview's branch refs/heads/2971-count-distinct from [~kocolosk] [https://git-wip-us.apache.org/repos/asf?p=couchdb-couch-mrview.git;h=f7c3c24] Add _distinct as a built-in reduce COUCHDB-2971
12. Commit 5d18415237e7a01e1ac401607f7fc36b671bf640 in couchdb-fabric's branch refs/heads/2971-count-distinct from [~kocolosk] [https://git-wip-us.apache.org/repos/asf?p=couchdb-fabric.git;h=5d18415] Add a finalize step after rereduce Currently this is a noop for every reduce function except the HLL cardinately estimator implemented in _distinct. COUCHDB-2971
13. Commit ee32cd5825aaf63448651c9521f0927083d2281e in couchdb-couch's branch refs/heads/2971-count-distinct from [~kocolosk] [https://git-wip-us.apache.org/repos/asf?p=couchdb-couch.git;h=ee32cd5] Add a cardinality estimator builtin reduce This introduces a _distinct builtin reduce function, which uses a HyperLogLog algorithm to estimate the number of distinct keys in the view index. The precision is currently fixed to 2^11 observables and therefore uses approximately 1.5 KB of memory. COUCHDB-2971
14. **body:** OK, at long last I've posted an initial pass at the implementation. Some noteworthy bits: - The precision is currently fixed to 11, so m=2048. This uses ~1.5 KB and has a relative error of about 2% - I simply called this reduce function _distinct, but I'm not sure that's the best name. Open to suggestions - I needed to introduce a {{finalize}} function in order to implement the reduce. We could use {{finalize}} to make some of the other builtins more efficient too by avoiding repeated conversions back and forth from epson. At some point in the future we could also think about directly exposing the finalize capability to custom aggregations in e.g. Mango. - If we do allow customized precisions, what should the API look like for that? E.g. do we need an "options" object as part of the view? {code} "views": { "cardinality": { "map" "...", "reduce": "_distinct", "options": { "precision": 12 } } } {code}
    label: code-design
15. **body:** Gave it try. Attached my rebar.config.script (the branch of couchdb-2971 in top repo was a bit outdated). Filled a db with 10k documents: {code} In [5]: db = s.create('db') In [6]: db['_design/dd1'] = {"views":{"v1":{"map":"function(doc){emit(doc._id, null) };", "reduce":"_distinct"}}} In [7]: for i in xrange(10000): db[str(i)] = {} {code} {code} http -b 'http://adm:pass@localhost:15984/db/_design/dd1/_view/v1' { "rows": [{ "key": null, "value": 9737.75978064411 }] } {code} About 2.6% difference as expected. Was curious what group=true output looked like: {code} http -b 'http://adm:pass@localhost:15984/db/_design/dd1/_view/v1?limit=2&group=true' { "rows": [{ "key": "0", "value": 1.0002442201269182 }, { "key": "1", "value": 1.0002442201269182 }] } {code} I like {{_distinct}} but wondering if they'd be confusion why it does not return exact results (like _count). It is a bit of a leaky abstraction since it exposes the precision trade-off to the user directly. Maybe {{_distinct_approx}}, {{_distinct_hll}}, {{_sketch_distinct}} (as in it's a sketch algorithm, maybe if we add count-min later)? Options API looks good. Not sure how hard would be to implement it. Would there be other uses for it? Maybe _stats or user views could find options useful as well. But having a simple version with a default precision might be a good start to see how people use it first. Noticed hyper code has some Erlang 18 only functions but looks like perf report code only, so no problem there. As Robert suggested to be consistent we might need to have couchdb-hyper version. Saw we do that for meck and other external projects. But agree if we can avoid that and for other deps would be less hassle. Was there some discussion about moving repos to GH anyway? Then forks will be more obvious.
    label: code-design
16. Commit 25bc44e805e2d685bee633ef9bfa833eda5b3796 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=25bc44e] Add hyper to the build COUCHDB-2971
17. Commit ebf808d3e4e462f46fb0af7dd453bc10668bdb04 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=ebf808d] Squash ee32cd58 to rebase COUCHDB-2971 work commit ee32cd5825aaf63448651c9521f0927083d2281e Author: Adam Kocoloski <kocolosk@apache.org> Date: Wed Mar 1 09:28:45 2017 -0500 Add a

cardinality estimator builtin reduce This introduces a _distinct builtin reduce function, which uses a HyperLogLog algorithm to estimate the number of distinct keys in the view index. The precision is currently fixed to 2^11 observables and therefore uses approximately 1.5 KB of memory. COUCHDB-2971

18. Commit ebf808d3e4e462f46fb0af7dd453bc10668bdb04 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=ebf808d] Squash ee32cd58 to rebase COUCHDB-2971 work commit ee32cd5825aaf63448651c9521f0927083d2281e Author: Adam Kocoloski <kocolosk@apache.org> Date: Wed Mar 1 09:28:45 2017 -0500 Add a cardinality estimator builtin reduce This introduces a _distinct builtin reduce function, which uses a HyperLogLog algorithm to estimate the number of distinct keys in the view index. The precision is currently fixed to 2^11 observables and therefore uses approximately 1.5 KB of memory. COUCHDB-2971
19. Commit 02f9c010f536f84939b664aa361016d3f32a0cec in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=02f9c01] Squash 5d184152 to rebase COUCHDB-2971 work commit 5d18415237e7a01e1ac401607f7fc36b671bf640 Author: Adam Kocoloski <kocolosk@apache.org> Date: Thu Apr 28 15:12:44 2016 -0400 Add a finalize step after rereduce Currently this is a noop for every reduce function except the HLL cardinately estimator implemented in _distinct. COUCHDB-2971
20. Commit 02f9c010f536f84939b664aa361016d3f32a0cec in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=02f9c01] Squash 5d184152 to rebase COUCHDB-2971 work commit 5d18415237e7a01e1ac401607f7fc36b671bf640 Author: Adam Kocoloski <kocolosk@apache.org> Date: Thu Apr 28 15:12:44 2016 -0400 Add a finalize step after rereduce Currently this is a noop for every reduce function except the HLL cardinately estimator implemented in _distinct. COUCHDB-2971
21. Commit 9122d458770d5df34bfdf3d8831600cebaa5e280 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=9122d45] Squash f7c3c24d to rebase COUCHDB-2971 work commit f7c3c24db17db5908894447e1822936212d61fcd Author: Adam Kocoloski <kocolosk@apache.org> Date: Wed Mar 1 10:37:00 2017 -0500 Add _distinct as a built-in reduce COUCHDB-2971
22. Commit 9122d458770d5df34bfdf3d8831600cebaa5e280 in couchdb's branch refs/heads/2971-count-distinct from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=9122d45] Squash f7c3c24d to rebase COUCHDB-2971 work commit f7c3c24db17db5908894447e1822936212d61fcd Author: Adam Kocoloski <kocolosk@apache.org> Date: Wed Mar 1 10:37:00 2017 -0500 Add _distinct as a built-in reduce COUCHDB-2971
23. I rebased this branch and did subtree merges to pull in the commits on the various repos that have since been merged into the main one. After looking around a bit I think this reducer should be renamed `_approx_count_distinct` as that seems to be an emerging standard. Examples include Google BigQuery: [https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#approx_count_distinct] MemSQL: [https://docs.memsql.com/sql-reference/v6.0/approx_count_distinct/] Oracle: [https://docs.oracle.com/database/121/SQLRF/functions013.htm#SQLRF56900] Apache Spark: [https://spark.apache.org/docs/2.3.0/api/java/org/apache/spark/sql/functions.html] I also noted that each of these implementations returns an integer as a response. I spot-checked Spark's implementation and confirmed that it rounds the estimate: [https://github.com/apache/spark/blob/1270e7/sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/util/HyperLogLogPlusPlusHelper.scala#L239] I propose we do the same. The other interesting observation is that BigQuery allows the user direct access to the sketch via `HLL_COUNT.INIT()` which allows computing a distinct count across datasets: [https://cloud.google.com/blog/big-data/2017/07/counting-uniques-faster-in-bigquery-with-hyperloglog] I can certainly see the use case there, and it would be technically easy to expose this, but without a ready-made function to merge the individual sketches I don't think it would get much adoption. So I'd say leave that out for now and keep things simple.
24. {quote}One caveat is that this reducer would not work for group_level reductions; it'd only give the correct result for the exact key. {quote} I'm not sure why I thought this caveat existed; the algorithm appears to work just fine with `group_level`. It reports the number of distinct keys that share the common group prefix in each row. I think perhaps I was saying that it would not be possible to compute the number of distinct *prefixes* at a given group_level.
25. Commit bac71a9cf808a09a8333afbe6e08dad55224511c in couchdb's branch refs/heads/add-js-tests-for-approx-count-distinct from [~vatamane] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=bac71a9] Add JS integration tests for _approx_count_distinct Jira: COUCHDB-2971
26. Commit 2dc733fb434982d27cde69b98cc990b77eeefdcc in couchdb's branch refs/heads/2971-count-distinct from [~vatamane] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=2dc733f] Add hyper directory to .gitingore file (#1365) Jira: COUCHDB-2971
27. Commit 6d44e17fccc44c377476247d9765fc573154097f in couchdb's branch refs/heads/master from [~kocolosk] [https://gitbox.apache.org/repos/asf?p=couchdb.git;h=6d44e17] Add _approx_count_distinct as a builtin reduce function (#1346) This introduces a new builtin reduce function, which uses a HyperLogLog algorithm to estimate the number of distinct keys in the view index. The precision is currently fixed to 2^11 observables andtherefore uses approximately 1.5 KB of memory. It also introduces a finalize step which can be used to improve the efficiency of other builtin reduce functions going forward. Closes COUCHDB-2971
28. Leaving a note here to attest that I considered the security and privacy implications of introducing this feature when I designed and implemented it. /me waves at prospective future auditor