Item 282
git_comments:

1. Unmarshal CacheLookupHttpConfig
2. Deallocate unmarshaled data
3. The look up parameters are intialized in the same as it is done in HttpSM::init(). Any changes there should come in here.
4. update the cache info config structure so that selection from alternates happens correctly.
5. ---------- * class CacheLookupHttpConfig *----------
6. used by ICP to bypass this function
7. in in in in
8. 'global' user agent flag (don't need to marshal/unmarshal)
9. this is a global CacheLookupHttpConfig used to bypass SelectFromAlternates

git_commits:

1. **summary:** TS-1919 Backing this change out, since it's been -1'd.
   message: TS-1919 Backing this change out, since it's been -1'd. git revert did not handle this nicely, since the trees have diverged so much. Instead, I simply made 5.0.x identical to current "master".

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
2. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just

using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

3. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
4. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
5. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
6. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates

modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

7. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
8. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
9. **summary:** Eliminate CacheLookupHttpConfig
   description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

10. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
11. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
12. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
    label: code-design
13. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just

using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

14. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
    label: code-design
15. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
16. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
17. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the

same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

18. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
19. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.
    label: code-design
20. **summary:** Eliminate CacheLookupHttpConfig
    description: We have a notion of creating (and transmitting) a very tiny subset of HttpConfigParams, in a struct named CacheLookupHttpConfig. As it turns out, this is generally not used, and as far as I can tell, cluster config provides the same / similar functionality (assuring that all nodes in the cluster uses the same config). One complication with this CacheLookupHttpConfig struct is that it sort of violates modularity, in that the I/O core, clustering and HTTPSM share this partial HTTP config in a non-opaque way. I have a patch that eliminates this (I'll post it later), but there are two thoughts / questions I'd like to discuss. 1) Do we feel it adequate to use the cluster config mechanism of distributing / sharing configurations across the cluster? Or do we really think that it's necessary to transfer the configs as part of every Cluster response message? 2) If we agree to eliminate the CacheLookupHttpConfig in favor of just using HttpConfigParam's (which are synchronized between cluster nodes), how important is it to preserve compatibility in the Cluster protocol? Right now, my patch does not do this (I'd break clustering between e.g. ATS v3.2 and ATS v3.4). For 2), we have a few options, the cleanest probably involves knowing the version of the Cluster message (does that exist today?). Before I go down that route, I'd like to ask the

people using clustering if they feel it important to retain compatibility such that you can run a cluster with a mix of v3.2 and v3.4 nodes.

jira_issues_comments:

1. If it breaks binary compatibility between clusters, I'd announce it in 3.4 and change it in 3.6 (or whatever the version after 3.4 will be).
2. It breaks compatibility "within" a cluster. Moving it out to v3.5.0.
3. I forgot, this also affects performance in non-clustering mode, since we avoid allocating and populating the CacheLookupHttpConfig on every request.
4. This is my current patch, but waiting until after v3.4.0 to land it (since it breaks cluster compatibility).
5. Moving to 4.2.0 as per https://cwiki.apache.org/confluence/display/TS/New+Release+Processes
6. Commit 812b668de63a387cabb0ff43e149817a5c79cb8a in branch refs/heads/5.0.x from [~zwoop] [https://git-wip-us.apache.org/repos/asf?p=trafficserver.git;h=812b668] Added TS-1919.
7. Commit 062789462cf34355627d35c8a2899d822e38950a in branch refs/heads/5.0.x from [~zwoop] [https://git-wip-us.apache.org/repos/asf?p=trafficserver.git;h=0627894] TS-1919 Eliminate CacheLookupHttpConfig This breaks compatibility with the cache clustering, so all nodes in a cluster would have to be upgraded at the same time. This eliminates an ugly passing of certain configurations through the cluster channels into the cache core. In addition, I think this can open up the possibility to make some cache related configurations overridable. That should be done as a different commit though.
8. This is now landed on the new 5.0.x "incompatible" branch.
9. re compatibility: same as cache, I'd say: upgrades been minor versions should be possible -- Sent from my phone. Please excuse my brevity.
10. Correct. That's why I landed this on the 5.0.x branch.
11. from the talking with Weijin, it seems the change will cause us some trouble, the CacheLookupHttpConfig is design to control the cache internal functions from the http side, due to that cache is a lower layer. the CacheLookupHttpConfig is there for some of the reasons: 1, in some situation, where some of the critical cache lookup or matching config may vary between every request or hostname: 1.1 you make a tiny change from the conf_remap plugin on that remap rule 1.2 other plugins that will use TSHttpTxnConfigIntSet() etc 2, when in cluster mode, the matching is done on the remote machine, and it is better to get that control message send with the request. from what we have learned from our usage, it is better to get more control over the remap point, to set seperated timeout, cache directives, matching directives etc. if we have something we can send to cache with remap plugin etc, that is far great. and that is the reason we refined the remap.config. so, basicly, the CacheLookupHttpConfig is a cache control message send from the http layer to cache, to avoid the reverse access of those informations, and get more flex on cache control. if we remove that, it will make trouble when we'd like to do more control over the cache behavior, we should supply some better solution on that.
12. **body:** Right, I definitely understand what the feature was there for. I think it possibly predates the concept of cluster configurations. Are you concerned that the configuration changes are not propagating fast enough and this would cause problems? Is it unreasonable to make it such that if you make such a dramatic change to your configs, that you have to do it while out of rotation? Also, none of these configurations are configurable through conf_remap anyways (I'm 99% sure, but I'll have to double check). In fact, part of my reasons for making this change is such that we can allow overridable cache configurations (that's not possible right now, because the cache doesn't have access to the general HTTP configs). - cache_global_user_agent_header(false), - cache_enable_default_vary_headers(false), - ignore_accept_mismatch(false), - ignore_accept_language_mismatch(false), - ignore_accept_encoding_mismatch(false), - ignore_accept_charset_mismatch(false), - cache_vary_default_text(NULL), cache_vary_default_images(NULL), cache_vary_default_other(NULL) Those are the configurations related to this, pretty sure none of these are overridable. The ignore_mismatch options are generally really bad to use, and you certainly couldn't safely use them anyways. The vary_default configs seems like a generally bad idea, but could be usable. Global user-agent maybe could be useful, but I doubt it really affect much caching at this point in time (but it *might*). The reason I'm asking is that, getting rid of this really would simplify the cache core quite a lot. ANd in fact, make things better. The configurations we'd "lose" are incredibly obscure IMO. If you feel that this feature must stay in the cache core, I'll back it out. I'd really ask you to take a second look though, before you -1 this. Thanks! -- leif
    label: code-design
13. Reopening.

14. **body:** I was looking at this some more, and yes, those things are not overridable (because of the fact that they are passed along in this damn struct :). With the patch e.g. {code} HttpTransactCache::calculate_quality_of_match(CacheLookupHttpConfig * http_config_param, // in HTTPHdr * client_request, // in HTTPHdr * obj_client_request, // in HTTPHdr * obj_origin_server_response // in) {code} would change such that all the cache calculating options can be overridable. In itself, I think this would be a huge win (you can change cache rules by remap rules). This would break if you have different configs / plugins on different machines, but that seems like a bad idea anyways. Is there anything we could do such that we e.g. know if two machines are out of sync on e.g. installed plugins / plugin versions, or remap.config lines ? The other benefit is that we would remove an entire allocator for every request by eliminating this. It gets allocated for every request, just to hold these "special" configs. If you feel this really must stay, I can look into making this records.config'urable, such that we pass CacheLookupHttpConfig's only when cache clustering is in effect. But then you would lose out on the benefits the changes adds, but maybe that's ok ? The reason this is coming up again is because I'm looking at e.g. the proxy.config.http.cache.ignore_accept_encoding_mismatch. I'm 99% certain that they were added and implemented wrongly, to solve a real problem. As such, I want to fix these, but to use them safely, I need them to be overridable configurations (via e..g conf_remap.so). Thoughts?
    label: code-design
15. Taobao people: What's the verdict here? If you feel strongly that passing those configs in cluster response messages is required, please let me know, and I will undo this commit.
16. Commit db67432fce4bc324856c17038269f327a9128efa in branch refs/heads/5.0.x from [~zwoop] [https://git-wip-us.apache.org/repos/asf?p=trafficserver.git;h=db67432] TS-1919 Backing this change out, since it's been -1'd. git revert did not handle this nicely, since the trees have diverged so much. Instead, I simply made 5.0.x identical to current "master".
17. uhm... doesn't that also "revert" some changes that [~amc] committed to 5.0.x?
18. This is blocked, for now at least, on the possibly removal of Clustering.