Experiments with Multi-Tenant Search using Bloomberg Clustered Private Cloud

Thursday, March 6, 2014

Bloomberg Clustered Private Cloud (BCPC) is an Infrastructure-as-a-Service (IaaS) platform based on OpenStack. In order to explore the use of BCPC for Bloomberg Vault, we built a prototype of its large-scale multi-tenant archive search on BCPC and measured the scalability and performance of the system.

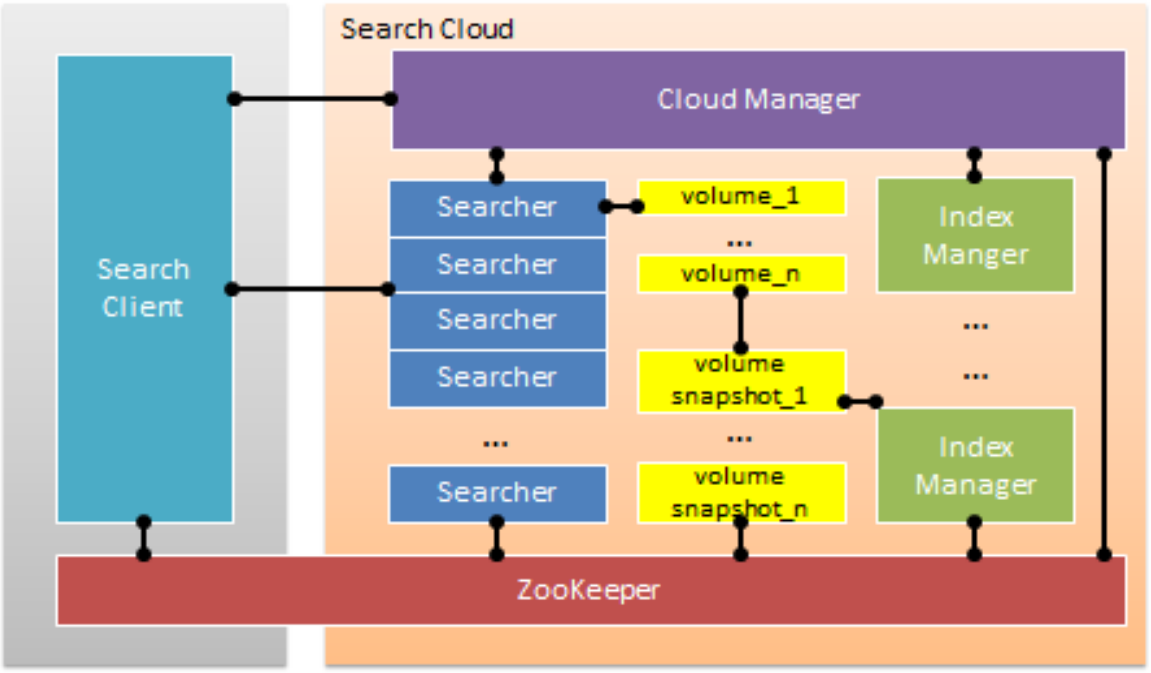
Bloomberg Vault is an enterprise information archiving service hosted within Bloomberg's geographically-distributed data centers. The secure, scalable and highly available service offers archive search, eDiscovery, real-time compliance monitoring and analytics to help customers manage their growing volumes of data.

Multi-tenant search in a cloud-based environment is a challenge because there is a need to make the trade-off between using a multiple per-customer index instance solution and comingling different customers' data into a single index instance. In the case of multiple per-customer index instances, frequent loading and unloading of different indices, as well as the spinning up of new processing nodes while the workload is already high, can have significant impact on overall performance. For a variety of reasons, our preference is for a multiple index solution, and this is what we configured and tested.

The primary goals for the prototype were to develop and validate our multi-tenant architecture using BCPC. Related to this, we had three additional goals:

- Use architecture patterns that are highly elastic in their ability to adapt to different search loads per tenant.
- Use open source components if feasible to reduce time to market for our prototype.
- Evaluate the performance of the search backend running on an OpenStack platform.

To start with, we present an overall architecture that shows the major components of the system:



At the core of the system are a set of components that interact with each other to coordinate the loading and unloading of resources and the execution of individual searches.

The **Cloud Manager** is responsible for managing resources in the search infrastructure including:

- Keeping the indices loaded on at least one Searcher node in the system. This ensures that the system does not spend time loading the index when a client wants to execute a query. Ultimately, if the combined indices are significantly larger than the amount of available system memory, as in our case, one must create a tiered architecture where the most important data is pre-loaded.
- Balancing workloads across Searcher nodes, and ensuring that there is enough headroom for Searcher nodes available in the system. The Manager can be configured to maintain a minimum number of Searcher nodes as a function of the size of indices that need to be pre-loaded.
- Repairing the cloud based on feedback from Search Clients (i.e. ensure fault tolerance locally). In normal circumstances, when nodes go down, Zookeeper should notify the Cloud Manager and a new node should be spawned. However, we built a secondary flow wherein a search client can provide feedback directly to the Cloud Manager whenever a search fails or exceeds its SLA so that it can rectify the problem as soon as possible.

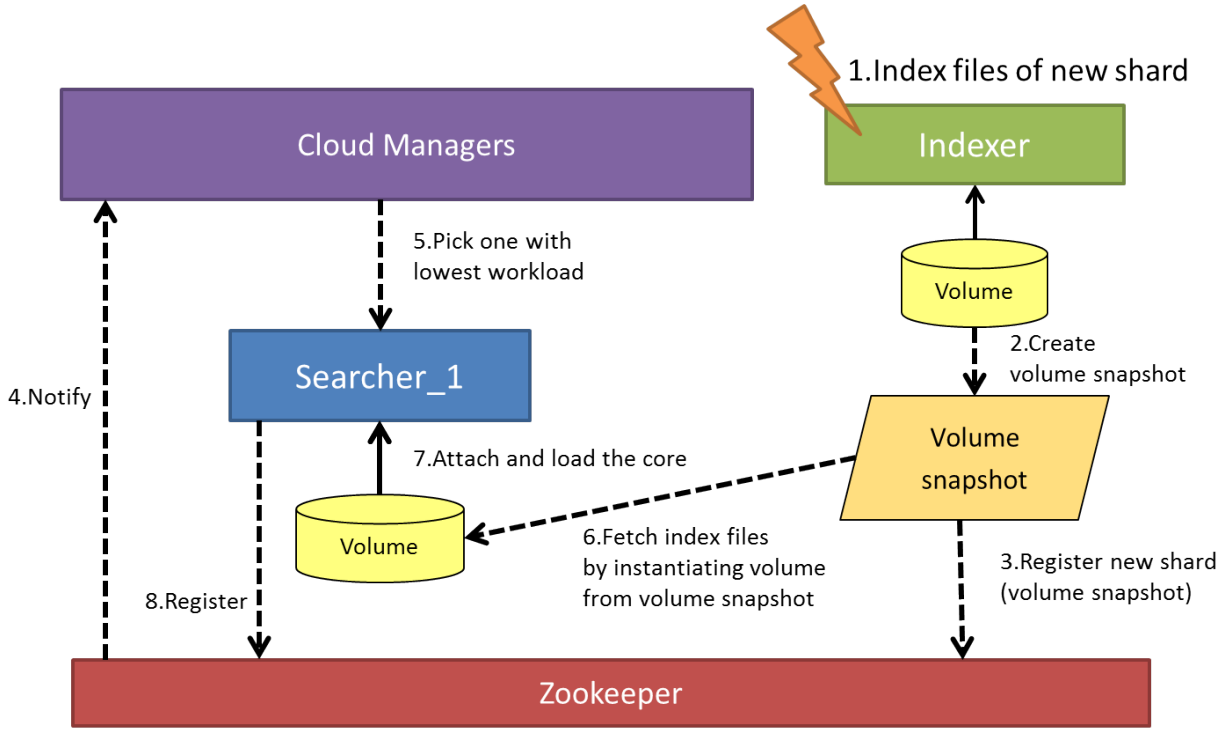
Searchers are compute nodes running Apache Solr. They mount one or more volumes with search indices and serve search requests based on each of the associated indices. The Search Client sends its query to one Searcher in the system, with a list of tuples containing a shard ID and corresponding node ID (both retrieved from Zookeeper), which has the relevant shard loaded. The Searcher receiving the request federates the search to all the nodes and is responsible for merging the results.

Index Managers handle new indices that are created by our indexing infrastructure and create volume snapshots for indices and move them into volumes. Future versions will move the indexing infrastructure into BCPC as well, but for the prototype, keeping indexing outside of BCPC helped us to prototype the architecture faster.

Zookeeper is an open source, centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. See Apache Zookeeper for more information. More specifically, this service performs the following functions in our architecture:

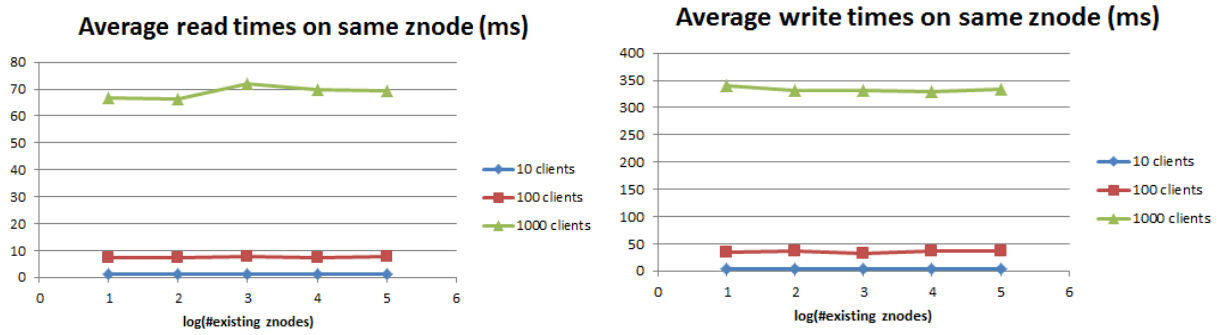
- Keeping track of and managing available nodes in the system (the spinning of nodes up and down depending on system load).
- Distributing change notifications for new indices, new/failed nodes, and search client feedback. (In particular, we use Zookeeper Watches to achieve this).
- Maintaining a repository of system meta-data and parameters (Indices, Searchers, Index Managers, etc.). This repository is used, for example, by Search Clients to determine which Searcher can handle a specific query.

The following figure depicts a basic interaction between the different components described above:

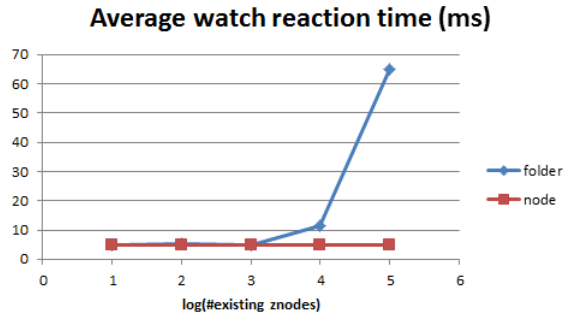


The process starts with a new index being created by the **Indexer** as well as a corresponding data volume. A **Volume Snapshot** is then created, registered with **Zookeeper** and made available to be picked up by a **Searcher**. The **Cloud Manager**, once notified by Zookeeper, will start a new Searcher that in turn will mount the Volume Snapshot and make it available for searching by customers.

One of the main concerns we had was the impact on read-write latency from Zookeeper as the number of znodes increase proportionally to the amount of data we store. Luckily, we found that read-write latency was only dependent on the number of clients connecting to Zookeeper and not the number of znodes in the system, as shown below:



We also found that the number of znodes did not impact the watch latency on any single znode. However, watch notifications on folders slowed down exponentially as the total file size of all the child nodes in the folder increased over 1 MB on a node with 8 GB memory, as shown below:



This is an acceptable trade-off for us, since the biggest Zookeeper folders in our system are ones keeping track of indices, which we always shard. As such, no single folder's watch time should grow beyond our performance needs.

Overall, our findings were as follows:

- Zookeeper and Cloud Manager working in conjunction provided a highly elastic foundation on which we could build our system. The Manager registers the znodes that keep track of system metrics, the number of nodes available to perform various tasks, as well as acting to provision new nodes, repair failed nodes, and repair issues with failed or slow searches.
- Leveraging open source products allowed us to build the complete stack in a couple of months with one full-time developer and an intern. BCPC uses OpenStack for its cloud platform and Ceph for distributing the data in the cluster.

- Initial performance numbers were very encouraging. A simple deep paging test fetching 10K results from Solr, with a 70 GB shard completed in half the time that it took in our current production system.

We plan to conduct a more detailed performance analysis and design description once the OpenStack-based architecture is fully built.