OmniTI

OAE technical infrastructure, application & operations

Table of Contents

1 Overview

As we noted at the end of the three-day "deep dive" exercise, the current paucity of performance test data limits OmniTI's ability to draw a definite plan of attack beyond the immediate first steps. Decisions pertaining to several potential key architectural changes (use of a graph database, use of a key-value store, reduce or eliminate reliance on aspects of SparseMap) are best made once we gather sufficient information about the application actual load and performance profile. Similarly, possible re-engineering options for the search architecture are best undertaken once we have a viable usage pattern to test the system against. Thus, the first necessary steps revolve around adding instrumentation and the ability to create a repeatably deployable environment against which a realistic load test scenario can be run in order to assess performance baseline and ongoing progress.

1.1 Document Scope

This document will detail and is limited to technical risks only.

Risk levels are: Low, Moderate, Severe and Critical and are relative to their impact on the system scalability and performance.

CTR stands for Cost to Rectify and represents an approximation of the level of effort necessary to implement.

Areas not covered:

- Disaster Recovery Planning
- Backups
- Fault-tolerant architecture

2 Risks

2.1 SparseMap

2.1.1 General concerns / maintainability

Issue: The OAE application makes extensive use of the SparseMap library. This is an open sourced library, created by the previous project lead architect, and it was designed to tackle core aspects of the system:

users and groups management, permissions management, replace Jackrabbit usage, search, etc. SparseMap is considered a high technical risk by the current OAE team. It is not a proven mature product and its design goals appear wider than the actual project needs; questions also exist regarding the level of external support that can be relied upon for maintaining and extending this component. The team has tended of late to treat SparseMap as a third-party dependency, and certain areas of the implementation have been treated mostly as a black box. This opaqueness however, does hide risks and it is recommended that the team gain a deeper understanding of SparseMap internals, should the project continue to use it.

Recommendation: Bring the SparseMap code inside the project. Consider reducing the overall dependency on the current SparseMap implementation by gradually replacing the various components used on the project with counterparts whose design goals are explicitly restated and match the current application requirements. More specific details are provided below.

Risk: Severe

CTR: Medium

2.1.2 JDBC layer with multiple back-end support

Issue: The flexibility coming from supporting multiple back-ends (Apache Derby, MySQL, PostgreSQL, MongoDB) is nice to have, but ultimately leads to unnecessary complexity and massively increased effort in testing, deployment and maintenance. Even during development, different environments use different back-ends making it difficult to uniformly evaluate the performance or effectiveness of specific features that are being implemented.

Recommendation: Retain the flexible design, but focus on a single back-end support per function, aka a single relational db, a single document store, a single graph db, should each of these become part of the final design.

Risk: Moderate

CTR: Low

2.1.3 API

Issue: OAE makes extensive use of the SparseMap provided API. Replacing SparseMap could be disruptive if the API changed.

Recommendation: The API mainly provides CRUD functionality (and Search which is actually not used). There is no need to change the API when changing the underlying implementation, except when additional APIs will be deemed necessary.

Risk: Low.

2.2 Data Representation

2.2.1 ACL Representation

Issue: Group memberships, document permissions are all stored relationally in an ACL table. This representation is not very well suited for ACL data. Querying this data requires multiple logical self-joins, mostly performed in the app, for performance reasons. Scaling such pattern in a relational database will be hard past a certain point.

Recommendation: Investigate the use of a graph database for storing ACL representation, such as Neo4j (http://neo4j.org/).

Risk: Severe.

CTR: High.

2.2.2 Content Representation

Issue: Representing sparse data in a relational db is suboptimal. If implemented directly, multiple properties lead to thousands of columns, most unused. If implementing a column database on top of the relational db, certain relational benefits are lost, as well as horizontal scaling ability.

Recommendation: Replace current content representation (cn_* tables) with a document/kv store. Document stores provide natural support for sparse data and make schema changes a non-event (only some app effort when changing property names). We recommend Voldemort (http://project-voldemort.com/) which brings several benefits to the table: Java app, handles high query volume, supports on-the-fly LZF compression, protobuf support, multiple serialization options, etc.

Risk: Moderate.

CTR: Moderate.

2.3 Caching

2.3.1 Ehcache

Issue: SparseMap uses Ehcache and each node in a cluster features several configurable caches (e.g., accessControlCache, authorizableCache, contentCache) that are replicated in a peer-to-peer manner using Ehcache's RMI replication feature. Since it runs per application server, its current use leads to cache-invalidation issues as well as cross-cluster priming issues. There also seems to be some level of uncertainty as to the proper functioning of this caching layer.

Recommendation: Allow configurable enabling/disabling of the Sparse level cache, to allow analyzing the flow without caching and then validate cache functionality. The cache should at least last for the duration of a request to avoid duplicate hits during same request. To avoid cache invalidation issues, set small expiration times (1-5s) based on app needs.

2.3.2 HTTP Caching

Issue: Numerous assets (images, scripts, style sheets, flash, etc) present on the page do not need to be reload it on subsequent page requests, or even between pages that share them.

Recommendation: Make sure Cache-Control and Expires headers are set appropriately.

2.3.3 Leverage a reverse proxy

Issue: Static assets, or rarely modified key-value data can benefit from being served via a cache/reverse proxy placed in front of the application servers. Additionally, dynamic connection management is better handled by such layer.

Recommendation: Run Apache Traffic Server in front of the application. Highly performant (200k reqs/second), employed by Yahoo and a number of other large deployments. If more flexibility is needed and SSL is handled elsewhere, Varnish could become an alternate option.

2.4 Search / Solr

2.4.1 Solr

Issue: For most current deployments, Solr runs under same JVM as the app. This both causes performance issues, as well as possibly hides data access issues behind the tight coupling.

Recommendation: Split Solr to run in its own JVM, even for test environments. All communication with Solr should be performed via HTTP.

Risk: Low

CTR: Low

2.4.2 ACLs in Solr

Issue: Access permissions for any content item are stored and indexed explicitly, flattened, for each piece of content. The application allows unbound sharing - a document can be shared with any number of individuals or groups, which could lead to requiring Solr to parse vectors of possibly thousands of groups, stressing Solr well beyond its common usage patterns.

Recommendation: Limit the number of entities a document can be shared with explicitly.

Risk: Moderate

CTR: Low

2.4.3 Search space size

Issue: Everything in the system is “search”. Any list displayed by the UI is generated behind the scenes via a search query. While only the current version of a document is indexed, there are no limits on the number of groups in the system, or the number of entities a document can be shared with (as mentioned above). Currently the search is performed as a Filter query based on the assumption/goal that the sharing would be predominantly public rather than private.

Recommendation: Some sensible limits should be enforced in the system as discussed. Using a Filter query won’t work well if sharing happens predominantly private, but this is a good approach to start with.

Recommendation: To cope with content growth, consider splitting search indices into several different layers. First is for newest documents (duration may be defined per document, depending on its purpose); this

index can be served by a system powerful enough to contain it in memory. Another layer would be the “archive” indexing old documents; an additional third layer could be dynamically generated for old but “hot” documents. Application UI would only search the new + hot layers by default, and will access the archival layer only with explicit user request.

Risk: Moderate

CTR: Medium

2.4.4 Date representation

Issue: Dates are currently stored as long data type.

Recommendation: Store dates as text using “YYYYMMDD” format. This enables prefix and range searches. All times should be UTC.

Risk: Low.

CTR: Low.

2.4.5 JSON Library

Issue: Current system uses JSON Lib to return data from search. This is not a streaming library, therefore large result sets would need to be loadable in memory.

Recommendation: Consider switching to Jackson (http://jackson.codehaus.org/) a streaming JSON library.

Risk: Low

CTR: Medium

2.5 Database Issues

These are issues related to the current use of a relational db. If these components are replaced with document store and graph db counterparts, some of the specific issues become moot. However, it is advisable that these issues are investigated, quantified and understood in the current format before architectural changes are undertaken, as the underlying behaviour will still be relevant and a performance baseline will be necessary in order to evaluate the success of taking a different approach.

2.5.1 Database Query Volume

Issue: current system exhibits a very high db query volume. Queries are simple (PK lookups) but there seems to be a lot of redundancy in data

retrieval (poor caching), no query aggregation (each PK retrieved individually), unnecessary queries due to selector parsing issue, etc.

Recommendation: Investigate current query patterns, verify query cache functionality and impact, eliminate unnecessary queries.

Risk: Moderate

CTR: Low

2.5.2 Data Growth

Issue: Authorizables and content tables suffer from potentially exponential growth given lack of bounds on number of groups that can be created and the document representation, including versioning, etc. Currently there are no good models for this growth and the behaviour is not fully understood.

Recommendation: Review data growth under a real load to better predict growth pattern. Explore which end-user limitations would provide relief without affecting user experience significantly.

Recommendation: If data remains in relational form, sharding will likely become necessary. For PostgreSQL look into using PL/proxy (http://pgfoundry.org/projects/plproxy/) (http://www.depesz.com/index.php/2011/12/02/the-secret-ingredient-in-the-webscale-sauce/)

Risk: Moderate

CTR: Medium

2.6 Instrumentation and Observability

2.6.1 Performance measurements

Issue: No instrumentation available that enables reviewing of timing/latency patterns for various operations in the application.

Recommendation: Wrap all potentially significant calls in timing events that can be enabled on-demand. For high transactional volume environments, statistical sampling may be a useful feature as well. Various frameworks are available in Java (see Resmon below) and others can be used for inspiration (see https://github.com/etsy/statsd)

Risk: Severe

CTR: Low

2.6.2 Trending

Issue: No trending data available to review data growth or application usage patterns.

Recommendation: Use Resmon (http://labs.omniti.com/labs/reconnoiter/browser/src/java/com/omniti/jezebel/Resmon.java) or any equivalent to instrument every point in the code or monitor any system metrics that can provide insight into the application functionality. API calls, events generated, Solr requests, data growth, etc. The instrumentation should be enabled as early as possible and data retained to review application behavioral changes with the introduction of new features, back-end changes or usage pattern changes.

Risk: Moderate

CTR: Medium

2.7 Deployment and Maintainability

2.7.1 Repeatable, automated deployment

Issue: There should be a method to deploy a full working cluster with basic configuration, with as little human interaction as possible. Sakai team is working on having all systems deployable under Puppet.

Recommendation: Continue the efforts to enable a Puppetized deployment. Additionally, review the possibility of creating pre-configured VMs that can be shipped to customers for fully controlled deployments.

Risk: Low

CTR: Medium

2.7.2 Upgrade/migration strategies

Issue: Currently, migrations require that every row is read and re-indexed, leading to large downtime.

Recommendations: Effort should be put in maintaining backwards compatibility; extra effort in the application layer is preferable to a painful migration which will push customers away from staying up to date. Make sure migration processes batch data loads properly and disable auxiliary events triggered during normal operation. The data export/load process can be performed offline, thus avoiding stress on existing system and sync complications.

2.7.3 Load Testing

Issue: There is no well defined method to pre-load a system with representative test data for performing repeatable, well-defined load and performance tests.

Recommendation: Develop a method to pre-load the system with test data and develop test scripts based on real-life access patterns observed in production deployments. The data load component could overlap/be reusable for upgrades/migrations.

Risk: Moderate

CTR: Medium

2.7.4 Release Cycle

Issue: Current product is moving from a yearly release cycle to several times a year. Slow release cycles have increased deployment and adoption risk due to a large number of new features that need to be tested and integrated.

Recommendation: Move towards a quasi-continuous deployment release mode with small changes available independently, as soon as possible. This is significantly easier if the system moves towards a single-hosted, software as a service model, but it can be achieved in standard software delivery model as well with a well defined painless upgrade system. Start exploring the use of dynamic feature flags and dark launches for significant changes to the system and A/B testing.

2.8 UI Optimizations

2.8.1 High number of HTTP requests

Issue: The app currently issues a high number of http requests.

Recommendation: Batching requests together, combining stylesheets and CSS resources, developing aggregate API calls will help reduce page load times and improve user experience. API call aggregation is particularly significant for mobile, but it is already understood/expected that the current front-end will not be the one used for mobile.

Risk: Low

CTR: Low

2.8.2 Set proper Expires headers

Issue: Expires headers are essential in controlling client-side caching behaviour.

Recommendation: Make sure proper expires headers are set, especially for hashed content.

2.8.3 Incorrect Counts display

Issue: Various pages display counts of the potential number of documents returned, which do not take into account additional filtering that will be applied after the fact.

Recommendation: To avoid user confusion either eliminate the counts or use vague qualifiers “viewing documents 20-30 of many”, similar to the GMail search feature.

Risk: Low

CTR: Low

2.9 Hardware

2.9.1 Storage Solution

Issue: Cost, performance and scaling capabilities should be considered when choosing between a SAN/NAS or DAS solution.

Recommendation: SANs are the historically preferred choice for enterprises, but in modern architectures, especially when storage requirements grow above the 50TB mark, cheaper, more flexible solutions are commonly employed. Direct attached storage solutions allow mixing and matching of different vendors, thus eliminating one inherent, often overlooked risk in SAN solutions (firmware bugs), and their low cost allows for a flexible choice of data redundancy for both reliability and performance.

2.10 Miscellaneous

2.10.1 Selectors

Issue: Current system allows the use of selectors to be appended to a document path using ‘.’. This causes Sling to apply trial-and-error search methods to determine the actual document path before treating the

remainder as selectors and applying them. This leads to a number of unnecessary DB queries that return no data.

Recommendation: Path determination should be deterministic. Implement selectors as query strings.

Risk: Low

CTR: Low

2.10.2 QoS

Issue: There is a Quality of Service layer in the existing system, but it mainly uses continuations to provide rate limiting at the wrong level in the app.

Recommendation: Proper quality of service requires application knowledge to be smart. Best methods rely on identifying different classes of customers and make sure they only a percentage of customers are declined access to the app or experience deterioration of service, while the others continue to operate properly (carrier approach). A more effective way to indirectly maintain good QoS through the system is to define acceptable response time limits for all the potentially long-running operations in the app and cancel the ones that go over the threshold. Ideally these are enforced as close to the source as possible rather than in the app (ie in Solr or the DB, etc.)

Risk: Low

CTR: Medium

2.10.3 Analytics

Issue: The business owners of the product will derive a lot of its value from deep introspection of usage and access patterns. Provide a platform that will enable them to perform such analysis.

Recommendation: While determining what the appropriate analytics platform would be is premature at this time, the system should be instrumented to record all user activity and log this information for later processing. Hadoop is the typical target for this kind of massive data dump.

2.10.4 Multi-tenancy

Issue: There is an expressed goal that the application should be deployable in a multi-tenancy environment for ease of use for institutions without the full resources required for running their own deployment. This has additional benefits for maintenance, as a single upgrade benefits all tenants. Conversely cross-tentant QoS and data access isolation become significant.

Recommendation: There is no silver bullet. Multi-tenancy should be an integral part of the application design, allowing for data partitioning, access isolation, etc.

Risk: Moderate

CTR: Medium

2.10.5 Cross Organization Content Sharing

Issue: It is a stated goal of the OAE product to enable and encourage data sharing across institutions. The success of this initiative may depend on the approach to facilitate it.

Recommendation: A truly distributed approach, involving sharing between separate deployments is an extremely challenging problem. A simpler approach may be sufficient though: since all data access is based on search, implement cross-institution search as a hierarchical access, triggered only by explicit user request. If documents are allowed to be "copied" cross-institution additional challenges ensue in terms of document owner/manager. Problems become potentially much simpler if we are looking at a single, multi-tenant deployment. At this point though, tackling these problems is premature.

Risk: Low

CTR: High

2.10.6 Chat

Issue: A chat manager API, service and chat servlet is provided in Nakamura. However, as it was causing significant load issues it has been disabled in current production deployments.

Recommendation: Chat should be completely decoupled from the web app. Given its significantly different access pattern, chat should be allowed to grow/decline independently of the application. Also, web sockets have gained full adoption and are the preferred method over Comet-based

implementations. A Flash plugin can handle backwards compatibility for older browsers.

2.10.7 Thread Safety

Issue: It is important, for scaling various components of the system, to verify that the app is thread-safe.

Recommendation: Verify application thread-safety

Risk: Moderate

CTR: Low

2.10.8 Compression

Issue: Documents should be stored compressed.

Recommendation: Use LZF/LZO compression as data goes into the document store. Some stores do it natively (Voldemort). File Systems support it as well. It is cheap.

Risk: Low

CTR: Low

2.10.9 Application goals

Issue: Various additional capabilities have been proposed for OAE, such as assessment, calendaring, chat, grading, etc.

Recommendation: Consider carefully the advantages/disadvantages of simply extending OAE "natively" by building into it services such as assessment, grading, calendaring, chat, etc. Requirements for the various applications should be determined on a case-by-case basis. Some may function effectively within the framework set for the entire system, while others may have more complex relational patterns better served by an external integration strategy, utilizing "stand-alone" services/apps built by the team or provided by third-party vendors.