A funny thing happened on the way to a billion...

Alfredo Alba, Varun Bhagwan, Mike Ching, Alex Cozzi, Raj Desai,
Daniel Gruhl, Kevin Haas, Linda Kato, Jeff Kusnitz, Bryan Langston,
Ferdy Nagy, Linda Nguyen, Jan Pieper, Savitha Srinivasan,
Anthony Stuart and Renjie Tang

IBM Almaden Research Center, San Jose, CA, USA

Abstract

IBM's Semantic Super Computing platform has been designed to ingest, augment, store, index and support queries on billions of documents. It faces a number of requirements and challenges not found in similar, smaller content management systems. We describe some of these challenges and lessons learned in the areas of solution design, hardware, operations, middleware, algorithms and testing.

1 Introduction

As document repositories approach the multiple billion document level, users are finding that simple search no longer meets all of their information retrieval requirements and traditional approaches to other content management functions are proving to be inadequate. As a result, research is focusing on new architectures and technologies that enable the development of the next generation of advanced information management applications.

Over the last 8 years we have built a Web-scale content management system [12, 8] to collect, analyze, store and serve billions of documents. In doing so, we have encountered a number of ways in which such systems must differ from traditional content management systems. Even though our initial focus was Web data, we've found that many companies have similarly large collections of data and information needs.

In this paper, we present several of our hard-won lessons for handling information storage, retrieval and analysis requirements at the multiple billion document level. These include new insights in the areas of information discovery, data sampling, performance management, system administration and operations.

2 System Considerations

2.1 Discovery is more than Search

Corporate document archives have rapidly moved to the multiple billion document level under the pressures of various regulatory and compliance rulings. As our clients begin to leverage increasingly valuable information out of these unstructured document archives, they are finding that simple search, while a valuable tool, is not

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

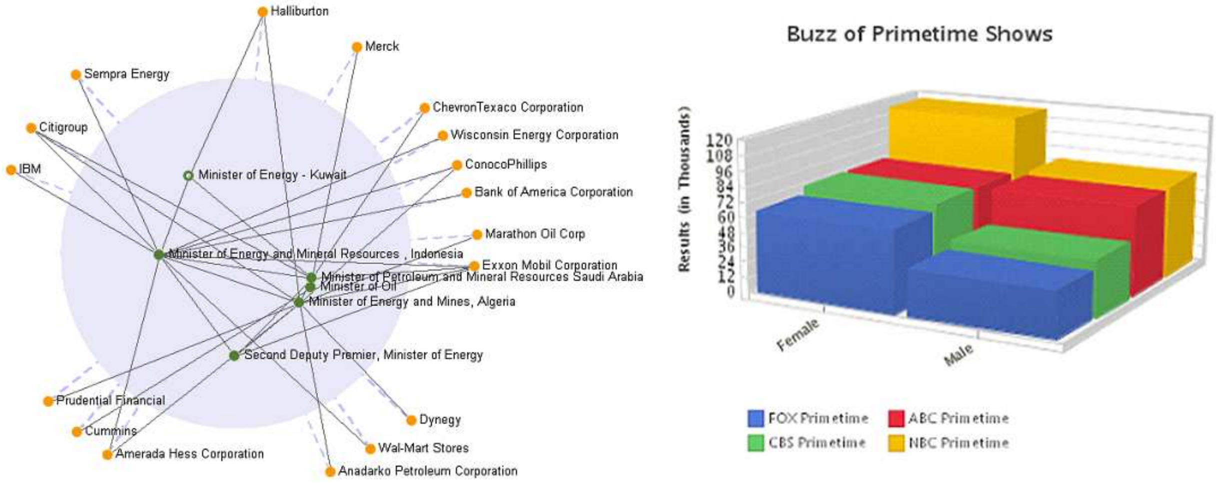

Figure 1: Discovery is more than search. An analyst may be interested in the names of companies most frequently mentioned with OPEC ministers, and the relationship between the ministers and the companies based on Internet data (left). Another analyst might be interested in the TV networks watched most by users of a popular on-line community site, broken down by male and female users (right). In both cases the information being sought is aggregated at a high level and presented in a business intelligence like manner - very unlike traditional search responses.

enough. Any one search request may match a very large number of documents. Simply listing the 150,000 hits is not an option. Analysts need tools similar to the "big picture" business intelligence reports common in the structured data domain.

We refer to such suites of tools as *Discovery*. Discovery applications support the production of charts, graphs, plots and other visualizations, as well as allowing drill-down to the source documents that support them (see Figure 1 for examples). They allow the definition of complex annotations and taxonomies, with subsequent application of such knowledge structures to arbitrary corpora. For example, if the corpus was a corporate email spool being examined for a compliance application, an annotator might be looking for something that could be construed as a stock tip. A pharmaceutical company might want to scour web data for potential patent violations. Discovery systems must support alerts or triggers, which run when new data enters the system or conditions change. Analysts need to be able to track a topic over time [14].

These more mature information needs impact the underlying infrastructure with a number of novel functional and non-functional requirements. There are certain query operators that are not common to the pure search domain, such as aggregation and reporting of accurate matching result cardinality (as opposed to the "estimates" that traditional search engines might provide – see Broder et. al. [5] for an example of the problems here). Since many of these plots require multiple (often thousands) of underlying "searches", the system has fairly rigorous performance requirements so as to be able to return the results to the user in seconds rather than hours.

Unstructured data requires the generation of metadata in documents to support general and task-specific queries. Most authors do not (and in many cases cannot) anticipate all of the tags that might be needed someday, and the size of the corpus makes manual tagging impossible. Robust automatic metadata annotation systems, such as UIMA [9, 11], are required to derive the higher-level concepts that need to be queried. In the example in Figure 1 (right), we are looking at "network prime time shows" mentioned on the web, even though that exact phrase probably does not occur. Instead, at ingestion time, an annotator examines each document and identifies shows that are being mentioned in any of a number of ways, and then annotates the document with metadata indicating which network the show is carried on.

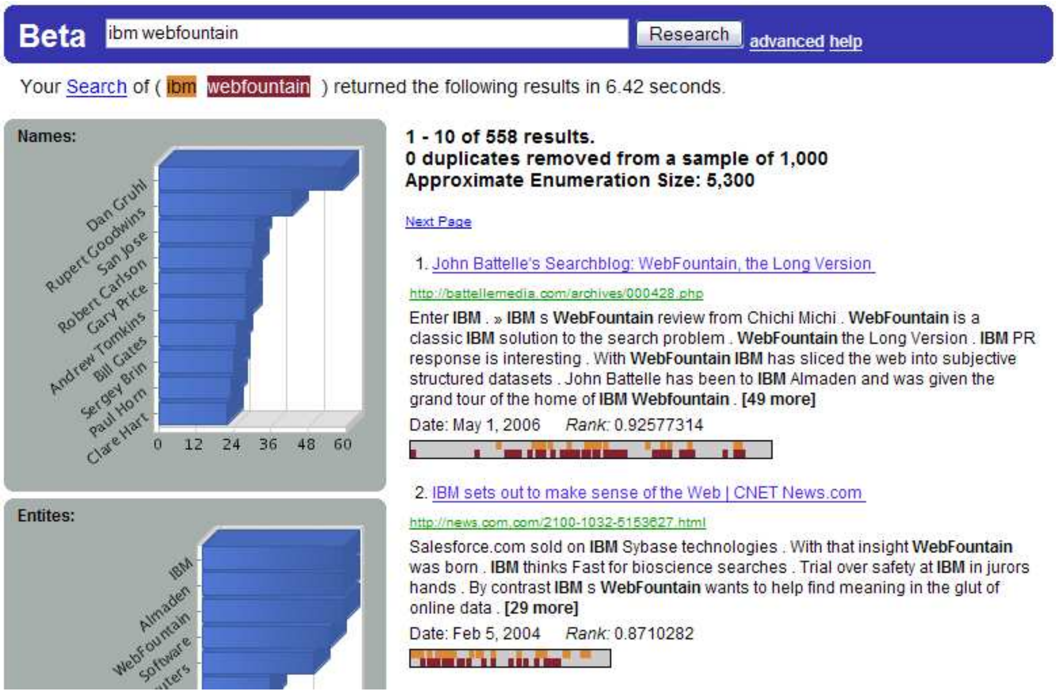

Figure 2: Enhanced search with additional result set statistics displayed in the sidebar.

Once the results are assembled, they need to be presented in a meaningful way. Such visualization also requires the ability to drill down to the supporting documents, as the underlying information may not be as accurate as is typical in more structured domains. Because of this, analysts find the ability to rapidly "spot check" certain results essential.

This is not to say that the system doesn't support keyword search queries. We've observed that most users begin to explore the system using keyword search, as it provides a familiar interface and a "feel" for the underlying data. Search therefore remains a core function of any unstructured management system and has to be done well.

Providing a reasonable level of abstraction between the applications and infrastructure requires a well defined API by which the applications can issue their questions and receive answers. This semi-structured query language allows traditional business intelligence style aggregation operators (e.g., COUNT, GROUPBY, UNION, INTERSECTION and their combinations) over lower level more search like subqueries such as ("IBM" or "I.B.M.") NEAR "DB2". Due to the complex nature of these queries we use a tree structure (i.e., XML) to represent queries rather than a left-to-right structure, but it otherwise the API strongly evidences its SQL roots.

2.2 People are not going to learn something new unless they have to

A user interface for discovering information buried within a large corpus of data poses challenges in itself, namely how the users interact with and drive the system. We have found that our clients are not naive users, thus a certain amount of analytic expertise and sophistication can be assumed. Nevertheless, a successful solution needs to be simple enough to provide a shallow learning curve, but powerful enough to allow experienced users to achieve maximum benefit. We are guided by the following principles:

- Don't overly complicate the system – it should be simple to start using the tools, even though the tools themselves may be quite complex. We shouldn't require the users to thoroughly understand everything before they can start achieving results. To quote Alan Kay, "Simple things should be simple, complex things should be possible."

- Employ paradigms users are familiar with – if the users just want to do a simple search, let them. Use familiar visual cues, such as search bars, graphs and navigation (links) between tools. It doesn't matter if a "better" approach is twice as fast if it takes years to learn how to use it.
- Interconnect as much as possible – relate functions together so that users can move between different tools/views of data. We provide applications that go beyond search. Combining search with other functions puts users in a familiar setting while augmenting the results with more detail, such as aggregate data. People understand search, but visualizing more information than just a result list can provide a fuller picture of the underlying data. For example, searching on a keyword returns a page showing not only a subset of matched documents, but also a distribution of the top `k` instances of various entities as found in the entire result set, e.g. names, email addresses, and dates. Users can then drill down or filter on any of these entities. See Figure 2 for an example of such a system.
- Grasping capabilities is not easy – users can start with search, and through it learn to harness more power of the analytics. However, this is a system for analysts – actually understanding the big picture in the data returned requires the experience to separate the valuable details from the chaff.
- Give users the power to narrow the corpus – allowing users to run analytics over a smaller set of documents lets them work within a manageable problem domain and reduces noise from irrelevant data. For example, users can reduce the scope by collections of sites; sets of entities such as documents containing names, emails or other sets; and other queries. This is especially useful when the users can create source collections of "authoritative" sources for a particular domain. Depending on the query, authoritative might be the MTV web site, or the American Heart Association – but few queries benefit from including both.

2.3 Sample early, sample often, and leave everything to chance

The benefits of sampling in traditional structured databases in the context of business intelligence type queries is well known. We have found that similar benefits can be derived from equivalent approaches in the unstructured content management domains that our clients are interested in.

If one has over 2 petabytes of data, streaming the entire corpus off disk and through the processors can take several days, even for a fairly large cluster. This means that for real-time query processing some kind of sampling is a must, especially for business intelligence style applications. Fortunately, statistical sampling provides significant performance advantages over traditional full-scan systems while still delivering highly accurate results.

Making a virtue out of this requirement, we have found that embracing random distribution simplifies and enhances many aspects of our platform. Documents are randomly distributed across the cluster by hashing a primary key (e.g. URL) to a cryptographically random 128 bit Unique Entity Identifier (UEID) and using the low order bits of this UEID to determine which node of the cluster to assign the document to.

This random assignment allows for the even distribution of documents across the cluster with minimal overhead, and makes it easy to determine where any given document resides on the cluster. A uniform and random distribution can be assumed, enabling analytics to pursue several approaches for random sampling.

In addition, the index on each node returns documents in UEID order (that is, fixed but random). Thus any sample out of an index is an unbiased sample of all the data on that node, which is an unbiased sample of all the data on the cluster. This allows queries such as "how many documents mention IBM in the same sentence as a database" to be processed through an estimator that converges like a standard polling sample. Typical 99% confidence level on a 50% true response from the entire population would have a margin of error:

$$MoE \approx 1.29/\sqrt{2n}$$

The factor of 2 is due to the index only returning positive hits (the negatives being the documents the index doesn't return). So if 10,000 samples are pulled (even off just a single node), the margin of error is 0.91% – probably good enough for most purposes. It should be noted that this serves as an upper bound. The actual MoE converges faster if the sample represents more or less than half the documents. Larger samples can easily be drawn nearly as quickly by consulting multiple nodes, but the MoE decreases non-linearly due to the square root. Unsurprisingly, it is much faster to pull 10,000 examples than it is to pull the entire 6 billion documents.

One example application which takes advantage of this sampling in our system allows users to search for the top websites (as opposed to web pages) discussing a specific query topic. Without storing website information directly in postings lists, this type of query would take prohibitively long without random sampling: it would be necessary to look up the hostname for each document hit, and aggregate across all matches. Random sampling allows us to examine a relatively small portion of the entire corpus, while retaining confidence in the accuracy of the result set.

New approaches to sample from a sorted index have recently been published in [5, 6]. However, the number of disk seeks required by these solutions is still orders of magnitude higher compared to using an index that already provides a uniform and random distribution. Most discovery applications rely on sampling to provide summaries of the underlying data. As such, we have found that an infrastructure that is optimized for these use cases is beneficial.

3 Hardware Observations

3.1 I/O is *the* bottleneck

With very large amounts of data it is almost always the case that the data an application is looking for is not in local memory – it either resides on local disk, or on a machine somewhere else on the local network. Any I/O operation goes through multiple layers: application call in user space, the file system or TCP stack in the kernel, and the network card or disk system in reality. Not surprisingly, with all these steps, there are very few analytics that do not end up bounded at some point by the amount of available I/O bandwidth.

For the disk subsystem, a sequential read performance on the order of 100 MB/sec is typical when using enterprise class drives and a RAID configuration. However, with discovery applications sequential access is rare. Much more common is randomly accessing many small files (on the order of 30kB each). Due to inefficiencies in every level of the I/O stack, a more realistic performance point for these smaller data units is 1-10 MB/sec, a drop of one or two orders of magnitude. This "narrow straw" to the data means that for semantic super computing systems handling billions of text documents, performance is often more limited by slow I/O than by lack of processing power.

One solution is to increase the number of "straws". This is one of the pressures that drives semantic super computing systems to scale out to large numbers of commodity systems: each provides one such straw. Taken together, a 256-node system can begin to process data at rates that allow reasonable performance at billions.

3.2 Newton's Law trumps Moore's Law

Why are the disk seek times so poor? Ultimately the culprit is that disk seek times have not shrunk nearly as fast as CPU speed has grown, because there are fundamental limitations on how fast a disk head can be accelerated and decelerated. This means that the time it takes to seek to a location on a disk and read a document (in this case around 30kB) quickly becomes a limiting factor. Making this even worse is that disk sizes are rapidly growing, thus leading to a tendency to reduce the number of disks that can be working in parallel.

Techniques like Asynchronous I/O (AIO) and Tagged Queuing emerging on SATA drives have helped, but ultimately some classes of problems are limited by this underlying seek time. This means that the most meaningful way to improve performance are optimizations throughout the I/O stack to limit seek counts. Additional strategies are to get more disks working on the problem in parallel through further distribution of the RAID arrays, optimization of allocation strategy, and the optimization of directory structure balancing algorithms.

Even with the directory tree traversal requiring multiple seeks, astonishing performance for both random reads and writes for file systems with up to one million files has been achieved. However, as one moves to billions of files, the overhead of managing thousands of separate file systems becomes prohibitive. We hope that in the near future, commercial file systems will be able to scale comfortably to over a billion files.

Until then, distribution of data across multiple file systems is the only solution. Taking advantage of distributed solutions leads to the question of how many nodes are required. Even at 256 nodes, each node needs to manage roughly four million files to reach a billion. At first glance, this seems to be well within the theoretical limit for the most commonly available file system implementations. In practice, as available space for allocation shrinks, a whole set of factors such as disk response time, RAID overhead, directory tree size and file system fragmentation can have a negative effect. The performance implications of storing well over 20 million files into a file system are significantly worse than initially anticipated. Overall performance deteriorates by more than 60% on average.

Some file system implementations deteriorate more rapidly than others. However, the trend is shared among all examined file systems, with significant performance degradation at volumes lower than 50% of their maximum theoretical limits. The good news is that off-the-shelf file systems can cope with millions of files in a single file system if the directory layout is carefully designed. The random reads required by discovery applications are more than an order of magnitude more costly than sequential reads. This single consideration seems to dominate effective read performance within our experiments. Intuitively, one would think that smaller directories with a three- or four-level deep directory structure would be optimal for such file systems as ReiserFS or XFS. However, this is not the case, as they often perform better when the millions of files are stored in a single directory. The results of our experiments suggest that systems need to be developed that perform well in all variations of the Create, Read, Update, Delete (CRUD) process, even when scaled to billions of files.

4 Operational Aspects

4.1 Cluster management by log file doesn't work

Like a high-performance race car, any large-scale computing environment requires a solid base supported by constant tuning. Spend too much time sifting through mountains of system and application details and key trends, alerts, cycles and patterns are missed. Typical IT shops achieve a server-to-admin ratio of about 20:1. To improve that ratio by an order of magnitude, one must develop and deploy the proper toolset that provides efficiency and automation. In our environment, we leverage a variety of complementary technologies and products that enable us to achieve desired uptime and monitoring goals, as well as drastically reduce the number of administrators required. The suite of technologies employed ranges from a proprietary cluster management solution to freely available open source toolsets such as Nagios [3], MRTG [2], System Imager [4], etc.

We established Nagios as our central repository for the critical system and operational data, then used its robust alerting engine to build proper rules. Using Nagios' SNMP capabilities, we configured our cluster to generate alerts on a variety of issues such as disk failures, RAID rebuild warnings, high environment temperatures, SSL certificate expiration, and more. Through the use of custom plug-ins, we extended the use of Nagios to monitor thousands of web services in our service oriented architecture.

MRTG's graphing engine is used to graph any numerical output. We use MRTG to visualize all real-time performance and operational status, whether it be SNMP or application-generated data. As problems occur, this technique reduces the time spent performing costly root cause analyses, and also serves as a key quality control tool. The ability to get high-level summaries of trends and issues is critical as examining low-level alerts from thousands of machines on a daily basis is laboriously expensive.

System Imager excels at deploying images to identical or similarly-configured hardware configurations. Using a static DHCP and PXE-based boot process, cluster installations for 1000 servers can be performed in under 12 hours by a single person. It also makes for an effective disaster recovery tool. We deployed two enterprise management tools created by IBM Research to improve our operational efficiencies. IBM's Cluster Management System launches and manages the thousands of web services in our cluster, organizing services and machines into logical groupings that enabled one-click operations. We also implemented BlueBoard [15], a touch screen-based tool that utilizes variable-length, multi-dimensional "star" icons to represent system or application status. An operator can spend a few minutes reviewing the cluster operations using this interactive tool, with problem details and closure being only a few finger taps away.

Finding the right toolset for a cluster environment is key. By reducing the time administrators spend collecting and analyzing an environment's system and operational data, they will have more time to fine-tune your high-performance race car.

4.2 SLAs are odd

Negotiating service level agreements with hosting centers for a semantic super computing cluster differs from the traditional data center expectations, where upwards of 99% server availability is the rule. Hosting a very large cluster under such a simple performance metric is not particularly meaningful, as well as cost-prohibitive. This kind of machine-level availability is required if the ability to query the data hinges on every machine being responsive. In such cases, given:

- A node count of N
- A node availability rate of $P(SA)$
- An assumption that node failures are independent

the probability of query availability ($P(QA)$) can be expressed by

$$P(QA) = P(SA)^N$$

Assuming a 256-node cluster with one machine per node, to achieve a typical data availability target of 95% would require a very expensive node availability target of 99.98%. Traditionally, this would often be implemented with redundant hardware, double-ended access to storage volumes, and other fairly expensive approaches.

Fortunately, due to sampling, our business intelligence style analytics do not require near-complete data availability to be able to support meaningful sampling and aggregation. As discussed in Section 2.3, we randomly distribute the data across the cluster, so sampling can be done on any available nodes without bias. For a given query, we merely need to identify how many nodes are needed to provide an estimate within the error bounds required of the sample.

We have found that for the kinds of questions our clients are interested in, solutions will perform well if at least 80% of the nodes are available. Thus, for our 256 node example we need:

$$P(QA) = \sum_{n=205}^{256} \binom{256}{n} P(SA)^n (1 - P(SA))^{(256-n)}$$

To achieve a data consistency service level of 95%, we only need a single-node service availability of 85% - a much more cost effective service point.

Why all this focus on SLAs and availability? With large systems, failures are not a risk but a certainty. Our super computing cluster experiences over 350 hard drive failures a year, 20 CPU failures, and numerous power supply failures, memory module failures, etc. The more flexibility available to address these problems, the lower the cost of doing so. While not quite "fail in place" [10], the 85% target provides us the ability to handle small and medium outages without degradation of data quality for our customers and their analytic applications.

5 Middleware Characteristics

5.1 Scatter/gather is non-trivial at 256 nodes and up

The promise of scale-out performance improvement is predicated on the ability to efficiently scatter queries to a cluster and gather the results back. Initially, seamless linear scaling can be achieved with simple solutions. As the degree of system scale-out increases, so does its complexity, both from hardware- and software-stack perspectives. Logically a client connects to a multiplexer component for a single entry point which hides the underlying large-scale complex system. Ideally, clients maintain a high degree of abstraction both at the logical and physical layers. The client should only be aware of very simple interaction parameters, which in this case are mostly time constants on query response. Reality though, is somewhat more complex. Consider what can go wrong when a query is sent out to 256 nodes:

- Some of the 256 nodes will almost certainly not be online (see Section 4.2).
- Of those online, a few might be either hung or responding very slowly due to hardware issues, machine load, periodic maintenance or upgrades, etc.
- The network switch or the multiplexer machine may become overwhelmed with traffic and start dropping or corrupting packets, run out of resources such as memory or socket descriptors.
- Mismatched network driver configurations may result in packet errors or performance degradation due to different MTU sizes or default duplex settings, which are not uniform in multi-vendor environments.
- Other issues include exceeding the maximum number of open file handles; inconsistencies within the system-wide configuration (kernel, memory, binaries, etc); and inconsistent access control configuration on runtime directories.

We have seen all of these problems and more. As a result, there is no such thing as a "maximum time" a query to the cluster may take, since at least one node will always time out. The multiplexer abstracts out this complexity, by hiding the detailed configuration and states of the system. It is required to handle error conditions, failures and partial failures of all related components. The multiplexer is also responsible for scattering the request to the individual components and gathering the results back. Each component executes independently, without any information about other nodes. Through the use of a highly optimized multiplexer, near-linear performance scaling has been achieved as the system scaled to the hundreds of nodes.

There is no single error-handling strategy that fulfills the needs of all clients. Instead, the state of each node is reported, so that the client can decide what the appropriate course of action is. The client can specify the mode of operation, which is used to tune the query execution.

6 Data Representation and Algorithmic Insights

6.1 One kilobyte per document over 6 billion documents is 6 terabytes

When dealing with very large numbers of documents it becomes necessary to pay a high degree of attention to the way storage trade-offs are handled that is quite foreign to today's programmers. Considerable thought must go into trading space for speed, data representations, and blob storage, or the system can quickly run out of space and/or get bogged down in I/O.

Semantic super computing systems constantly need to balance CPU usage against storage usage. Disk space can be preserved by compressing data and metadata or by generating metadata on demand – trading additional CPU overhead for less storage and I/O. Conversely, if the system has to respond as fast as possible, storing everything uncompressed may be the best solution. What to store and how to store it gains in importance, as the size of the corpus grows. One additional kilobyte per document over one billion documents requires an extra terabyte of storage. Furthermore, a larger corpus complicates the migration from one data format to another. Analyzing the storage footprint of an annotator and peer-reviewing the chosen data format becomes increasingly important.

Finding a good trade-off between CPU and storage can be tricky. We found that the best solutions are often counter-intuitive. In semantic applications that focus on unstructured textual information, standard compression using `gzip`[7] provides an excellent level of space savings at a reasonable performance cost. We apply `gzip` to the whole document and all metadata before committing it to disk. One surprising insight is that human-readable formats can outperform "hand tuned" binary formats after compression [13]. For example, representing an array of numbers as a string of white space separated values yields same or better data size after compression compared to a binary representation using BER encoding[1]. In addition, human readability clearly benefits developers as it simplifies debugging.

Another finding is that the use of Base64 encoding (a typical approach for placing "blobs" in XML) has surprisingly severe CPU and storage overheads. Some of this is obvious – Base64-encoded documents are roughly 30% larger than the original content, but there is a more insidious effect when compression is considered: the encoded content compresses quite poorly because a repeated sequence of bytes in the original document may be encoded in three different ways depending on the alignment in the data string, resulting in a potentially large decrease in compression efficiency.

Most data structures in our system contain large amounts of textual data, augmented with some binary data. One example is tokenized content, which contains each token (string), an offset (integer) and a length (integer). We found that converting the binary integer data to text results in better performance, lower storage requirements and easier debugging, than serializing such a data structure into a binary representation.

6.2 The need for speed, and 5% is only 1 part in 20

As is the case with any project that has a strong research component, there is always a desire to improve the algorithms used. But because we deal with such large volumes of data, this desire for increased accuracy must be balanced against the net performance impact when applying the algorithm across billions of documents.

Consider an algorithm that has been improved to produce results which are 5% more accurate than its predecessor. If one is given an infinite amount of time, and/or infinite computing resources, then the algorithm could be applied to a complete corpus and would result in an improvement in accuracy.

But in the absence of infinite time or computing resources, trade-offs need to be made. If the improved algorithm is quite a bit slower than the older algorithm, then it will process fewer samples in its allotted time. Fewer samples would likely produce overall *less* accurate results, even after taking into account the improved precision.

Due to the use of sampling for much of our work there is one aspect where it is almost always worthwhile to strive for improved algorithms: the removal of bias. Algorithms that introduce systematic bias or errors will produce bad results no matter how many samples are used.

7 Testing

7.1 When testing in the billions, "good enough" is not good enough

According to Murphy's law, whatever can go wrong will go wrong. This is especially applicable when designing a solution that is meant to process billions of documents. Consider the implementation of an efficient tokenizer, which needs to process thousands of pages per second. To avoid costly memory allocations, it may assume that no token is larger than 32 bytes, and pre-allocate memory accordingly. This implementation may work on a test run of a million documents, but will invariably encounter problems when applied to billion document corpora. For example, long concatenated nouns are common in German. Moreover, some data sources provide lower-quality data, such as Web pages with thousands of words and not a single white-space separator.

The above example may seem trivial, but when architecting a robust analytics platform that leverages in-house and third-party annotators [9, 11], and needs to survive whatever oddity it encounters, the only way to cope is by defensive design. No amount of testing can cover all scenarios that may eventually occur. The system must be able to gracefully handle crashing annotators without requiring manual intervention.

On the other hand, too much self-healing may degrade overall performance, necessitating the termination of a defective annotator. Unfortunately, most complex annotations depend on a chain of annotators that work in concert, and having one of them ejected can produce unpredictable results. Once again, there is no perfect solution. Just because a component was tested against a million documents, doesn't mean that it will survive the rigors of a billion pages.

8 Conclusions

In this paper, we presented our hard-won lessons for handling information storage, retrieval and analysis requirements at the multiple billion document level. We showed how, at this level, some of the traditional approaches to information management no longer apply, and how some of the best approaches may initially seem counter-intuitive. We hope that our experience proves useful for those designing their own large scale document systems.

References

[1] Basic encoding rules. `http://en.wikipedia.org/wiki/Basic_Encoding_Rules`.

[2] Mrtg. `http://www.mrtg.com`.

[3] Nagios. `http://www.nagios.org`.

[4] System imager. `http://www.systemimager.org`.

[5] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 245–256, New York, NY, USA, 2005. ACM Press.

[6] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 367–376, New York, NY, USA, 2006. ACM Press.

[7] P. Deutsch. Gzip file format specification version 4.3. RFC 1952, 1996.

[8] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW*, pages 178–186, 2003.

[9] D. Ferrucci and A. Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, 2004.

[10] C. Fleiner, R. B. Garner, J. L. Hafner, K. K. Rao, D. R. Kenchammana-Hosekote, W. W. Wilcke, and J. S. Glider. Reliability of modular mesh-connected intelligent storage brick systems. *IBM Journal of Research and Development*, 50(2-3):199–208, 2006.

[11] T. Götz and O. Suhre. Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43(3):476–489, 2004.

[12] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[13] D. Gruhl, D. N. Meredith, and J. Pieper. A case study on alternate representations of data structures in xml. In A. Wiley and P. R. King, editors, *ACM Symposium on Document Engineering*, pages 217–219. ACM, 2005.

[14] D. Gruhl, D. N. Meredith, J. Pieper, A. Cozzi, and S. Dill. The web beyond popularity: a really simple system for web scale rss. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 183–192. ACM, 2006.

[15] D. M. Russell, A. Dieberger, V. Bhagwan, and D. Gruhl. Large visualizations for system monitoring of complex, heterogeneous systems. In *INTERACT*, pages 938–941, 2005.