MONGOOSE: MONitoring Global Online Opinions via Semantic Extraction

Varun Bhagwan, Tyrone Grandison, Alfredo Alba, Daniel Gruhl, Jan Pieper
IBM Almaden Research Center
650 Harry Road
San Jose, California 95120 USA
{vbhagwan, tyroneg, aalba, dgruhl, jhpieper}@us.ibm.com

Abstract

The ever increasing amount of content on the Internet has fostered many efforts seeking to leverage this potentially yottascale information source. Service systems using advanced data and text analytics techniques have been developed to perform knowledge gathering and information discovery over Web data. Information gathered from free and public sources on the Web is frequently integrated with enterprise and proprietary data to create sophisticated service systems able to provide insight in an increasing number of business critical areas. Unfortunately, for fixed and or limited resource projects, consistent and reliable ingestion and integration of content often dominates the effort, reducing the time available for developing core analytics and presentations that differentiate and define an information service. If this initial data extraction, translation and loading of information (known as ETL in the database world) can be abstracted for these web sources, it would provide an important core technology on which Web-based information services could be more rapidly and inexpensively developed and deployed. This paper presents such a system – MONGOOSE – an approach that seeks to reduce the time spent creating a reliable data ingest and integration system and thus reducing the time-to-impact of advanced analytics service solutions.

1. Introduction

Service systems that perform data analytics are in high demand today; as people try to leverage existing information resources, whether in their organizations or freely and publicly available. The US Federal government spending alone on data analytics systems now totals in the hundreds of millions and does not appear to be shrinking [16].

Analytics efforts seek to extract interesting and often hidden "nuggets" from within their data by both looking more deeply at each source document, as well as looking across all data. In order to do so, however, all of these systems have to spend a vast amount of time, effort and resources simply acquiring, translating and integrating content. Thus, the focus of these efforts tends to shift away from data analysis – their stated core value proposition – and towards data ingestion, preparation and integration. In this paper, we present an information-seeking support system – MONGOOSE (MONitoring Global Online Opinions via Semantic Extraction) – that helps to address this imbalance. It should be noted that although MONGOOSE was created to fulfill the needs observed in our experience with web-scale analytics service systems such as WebFountain [8], SoundIndex [10] etc., it has also been successfully applied to diverse non-web content domains such as multi-modal mining for healthcare support systems [13], corporate communications analytics, and social network analysis. In addition to showcasing the broad applicability of an unstructured ETL tool such as MONGOOSE, these deployments demonstrate the plug-and-play aspect that allows system creators to rapidly adopt and adapt specific components of MONGOOSE to deliver complete systems to customers in previously unprecedented timeframes.

This paper is organized as follows. Section 2 presents an overview of the MONGOOSE system. Details of the system are presented in Section 3, with sub-sections 3.1 and 3.2 discussing its two core components. Section 4 highlights the key contributions of MONGOOSE, while Section 5 points to related work in this area. Deployments of MONGOOSE are mentioned in Section 6, and the conclusion of the paper and planned future work are presented in Section 7.

2. System Overview

At the most basic level, MONGOOSE is a software library with supporting code for control flow monitoring, analysis and correction that enables Worst-Case Scenario Workflow Management[1]. It allows community-based information extraction around specific phenomena that can be fed into statistical analysis tools.

[1] Worst-Case Scenario Workflow Management systems have Murphy's Law ("anything that can go wrong will go wrong") as their core principle and aim to mitigate these non-positive episodes.

The base MONGOOSE software package consists of the Intake Platform and the Control Platform (Figure 1). The user of this system can leverage a series of MONGOOSE modules to determine the particular information flow for the application they are developing. The user also tells the MONGOOSE system the form and content of the data they wish to have output. In leveraging the MONGOOSE modules, the user gets a resilient, fault-aware platform that ingests data irrespective of what happens either with the data sources or with the processing chain. This also allows MONGOOSE instances to be created for a multitude of domains; as instantiation involves plugging in domain knowledge cartridges into the system and then setting the outputs to a form suitable for the domain, e.g. OLAP or BI consumption. At the highest level, MONGOOSE functions as a robust *get* statement that recovers from failures. Activity is monitored by the MONGOOSE Control Platform, which analyses the system and source state; taking corrective action as necessary.

The MONGOOSE Intake Platform (Figure 2), hereafter referred to as the MIP, is comprised of software modules that falls into one of five categories: Acquisition, Pre-Processing, Language, Application Descriptors and Adjudication. As a large proportion of valuable online data is in "Broken English" [2], the system includes new techniques for spotting and making sense of "Broken English" use.

[2] Current analytics systems assume structured English and that Natural Language Processing (NLP) can be used. This is not true online.

The MONGOOSE base package contains acquisition templates for the automotive and music industries, pre-processing modules for comments and counts, the IBM Tunable Ranked List Producer algorithm for adjudication [2] and a Nagios-based [9] implementation of the MONGOOSE Control Platform, which performs workflow analysis and general system monitoring.

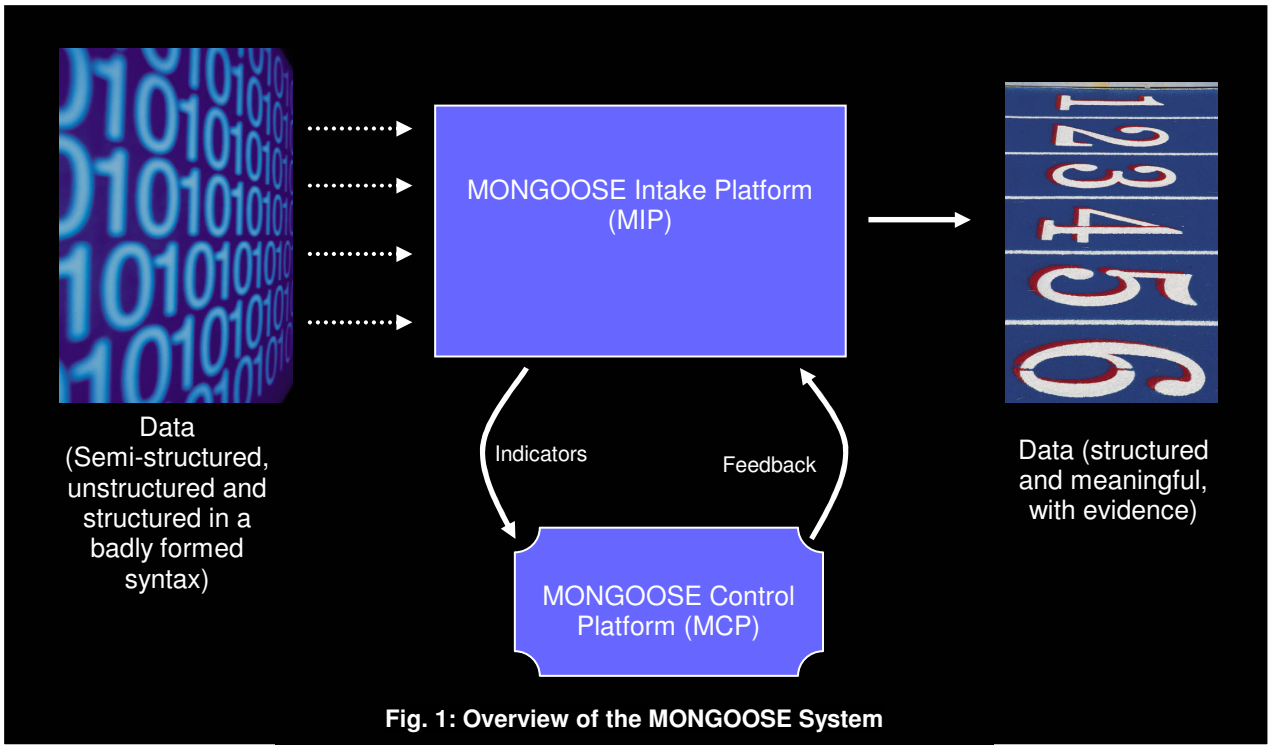

Fig. 1: Overview of the MONGOOSE System

3. Architecture

3.1. The Mongoose Intake Platform (MIP)

The MONGOOSE Intake Acquisition Modules (MIAMs) are base constructs used to build specialized ingestors for the domain of interest. The MONGOOSE Intake Pre-Processing Modules (MIPMs) extract individual comments, posts, discussion points, profiles and counts from the ingested data and processes the unstructured content to determine spam and identify on-topic and off-topic information. The MONGOOSE Intake Language Modules (MILMs) allows the jargon of the domain of interest to be included. Dictionaries for terminology in various domains such as healthcare, media and entertainment, law, automobiles, politics, etc. are included in these modules. The MONGOOSE Intake Application Descriptors Modules (MIADMs) allows the definition of the key descriptors for the domain and problem of interest. For example, in politics underlying concepts would be Integrity, Record, and possibly Funds. The MONGOOSE Intake Adjudication Modules (MAMs) combine multi-

modal information in a way that is meaningful for the business task at hand.

3.1.1. Acquisition. The basic acquisition of the various artifacts is usually conceptually simple but practically quite difficult. This is often simply issuing an HTTP GET command and passing the result on to the preprocessing stage. The complexity arises when such a command fails. Depending on the source, this might require attempting to reconnect to the server, flushing DNS caches and seeing if the server being used has been taken out of service, tracerouting to see where the problem is occurring and possibly trying alternate network connections that might bypass an outage, etc. When the source is not a simple page fetched via HTTP the problems become even more complicated.

3.1.2. Preprocessing. The page or data artifact produced by the acquisition stage is often a combination of the data we would like to get and other non-interesting data (such as ads, template content etc.). There is often markup that produces a visually interesting page that needs to be removed for semantic processing. Some pages include multiple low level artifacts – for example a bulletin board that includes multiple posts on a single page, or a review which may consist of a single artifact spanning multiple pages.

This is also the stage where preliminary, often heuristic spam removal can occur – for example repeated posts of identical comment or posts that follow common spam pattern (e.g., "check out URL"). At the end of this stage we have a collection of artifacts ready for more complex processing.

3.1.3. Language. Every domain has its own linguistic peculiarities. From a medical report that uses odd abbreviations and jargon to a social networking site where posts use odd abbreviations and jargon, the system needs to "transliterate" the artifacts into unambiguous English. This often includes expanding product names, slang, abbreviations, etc. and tagging out relevant entities. Basic actor/action/object parsing may occur at this point as well. These "parsed" artifacts with all of their meta-data are then passed on to the application module.

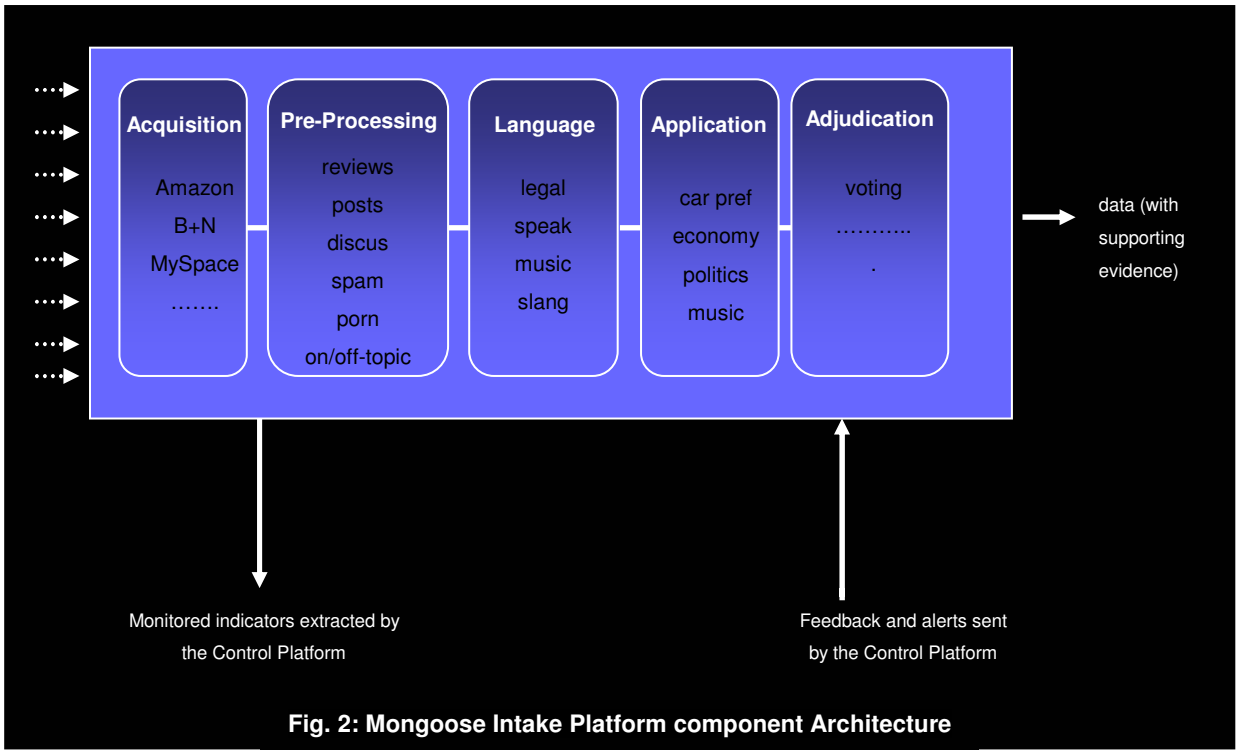

Fig. 2: Mongoose Intake Platform component Architecture

3.1.4. Application. Different applications care about different aspects of the source data. A music popularity application would be interested in artists, albums and tracks; specifically what people have to say about them. A car "buzz" application might be interested in models, statistics of the cars and tracks of an entirely different nature.

It is at this stage that the unstructured and semi-structured posts are converted to "dimensions" of structured information – for example the number of doors on a car, the frequency with which a particular musician is being discussed, etc. These vectors of data are then passed on to the adjudication stage.

3.1.5. Adjudication. One problem with dealing with opinions is that people have a tendency to disagree. Thus the vectors generated from the application section may contain contradictory opinions – one person might find the car sporty and fun and another might find the same car

cramped and difficult to drive. The adjudication module needs to combine these vectors together to create a coherent view of the underlying data. This might be as simple as counting the number of positive and negative comments on a discussion board, but may become more complicated when the underlying data comes from multiple sources of very different modalities (sales of songs verses listens of songs versus views of music videos). Some decisions need to be made how to do this combination that makes sense for the application (please see [2] for an extended discussion of the application of voting theory to this space).

3.2. The Mongoose Control Platform (MCP)

The MONGOOSE Control Platform (MCP), allows system continuity without visible failure (Figure 3). The purpose of the MCP is to reduce the Total Cost of Ownership (TCO) of systems both as they are developed and after they transfer either from prototype to development or from research to production. The MCP accomplishes this by providing the following general functionality:

1. Standard error detection, handling and reporting
2. Standard data analytics on corruption and consistency reporting, data acquisition layer execution state
3. Data flow integrity and source accessibility assertion
4. System wide data volume monitoring to ensure an increasingly monotonically increasing input data stream.
5. System wide data flow and cube output volume monitoring.

The MCP architecture depicted in the diagram above comprises the necessary components to fully assert the functional state of the underlying data flow.

3.2.1. Control Data Acquisition. It receives and processes protocol related error codes at all levels from HTTP fetcher error codes to data source response times including socket timeouts, etc. The Control Data Acquisition components also gathers and parses Semantic Extractors Failure Codes such as pages missing, error pages with valid protocol codes, format changes related to screen scrapers' page content changes or API's (at times unannounced) evolutions for instance.

Additionally, it is also responsible for handling all data processing error codes such as aggregation errors, unexpected data type errors, data output integrity checks, data corruption error codes and data consistency on a per modality basis including monotonic and non monotonic modalities.

When an exception is detected the MCP generates the corresponding event. All events are forwarded to the Data Consistency Collator for further evaluation, classification, corrective action execution, if applicable, and storage.

3.2.2. Data Consistency Collator. This component stores all data pertinent to the platform's execution state assertion. It associates source accessibility with the platform's state assertion for example website, network and gateway outages, etc. It dictates, and manages the standard reporting format for acquired data volumes, output data volumes, and output integrity checks in order to detect duplicate and redundant acquisition.

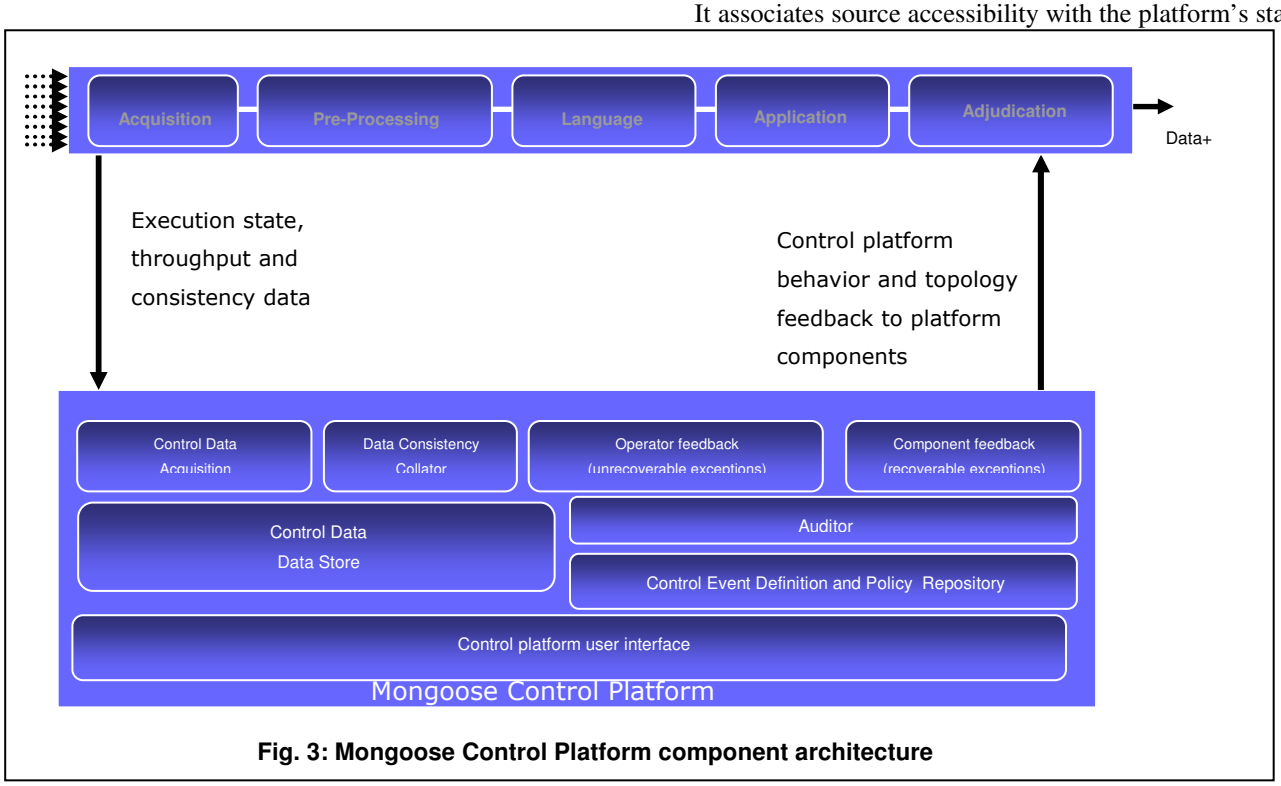

Fig. 3: Mongoose Control Platform component architecture

This is also the component responsible for maintaining the overall platform's functional states, as well as data throughput on a per-processing unit basis.

Overall the data volumes at this stage are increasingly monotonic processing unit attributes. Because of this, the analytics processing capacity high watermark is also maintained within the component at this stage.

3.2.3. Operator Feedback. Unrecoverable exceptions do happen, they simply are a fact of life on regular data acquisition operations, as internet sites and web services become unavailable, experience outages, credentials expire, or simply site operators decide to deploy unannounced evolutions, therefore effectively making previous versions unavailable.

In this environment operator feedback is vital. Once the Control Data Acquisition component has generated an event and it has been classified as unrecoverable an Operator feedback message is produced calling for operator action to be taken. The message and all its attributes are, of course, subject to the system policies as defined in the MCP policy repository.

3.2.4. Component Feedback. Recoverable exceptions are frequently observed as well. Spurious short lived outages, site response time slow down, data delivery delays because of unexpected increases in data volumes returned, are some examples of what could be considered recoverable exceptions, and handled through dynamic reconfiguration of the affected components and its dependants.

In this case all events are logged to allow for operator manual audits, while at the same time recovery actions are applied based on the defined policies. The actions are dynamic changes performed to the appropriate acquisition components and its dependents by the platform. In this particular case no manual intervention is required.

It should be noted that the there are significant Total Cost of Ownership (TCO) savings to be realized from the adaptive nature of platform's MCP capabilities.

3.2.5. Auditor. This component continuously verifies expected data volumes per source over the defined time windows. It compares the current data volumes against the expected values given the current platform's functional state, logs the results and decides whether a platform event needs to be generated at any given point in time. This is a vital activity necessary to assert the platform's data flow state and output data quality.

The component continuously performs data consistency checks on expected modalities on a per source basis, it tracks dataflow volume across sources and cube generators, as well as the general data cubes scalability and flow.

An integral part of the component also involves data consistency checks on all post processed data.

The generated events undergo the standard event path and are acted upon based on the same policies as any other exception event.

3.2.6. Control Event Definition and Policy Repository. This component collects and stores all event definitions, their category severity, priority, etc. Another vital function of the component is related to the policies associated with platform events. The event policies include actions, conflict resolution for individual and chain of events.

The defined actions can be component configuration changes consumable by the component feedback module, or operator suggestions regarding the current event.

Once an event is detected, it is classified, stored and the corresponding policy(s) applied at once, by the MCP components acting on the affected system components or issuing an operator notification.

3.2.7. Control Platform User Interface. The control platform user interface implements an human interface for MCP management, Essentially implements the platform console on one hand with on demand a data quality audit capabilities. It provides the ability to view and modify policies, perform event management, and general audit activities related to the platforms execution state. Most importantly it also provides audit capabilities on the output data quality of the current platform deployment.

It also delivers aggregated data views on all aspects of the platform's integrity. For instance all component feedback events as well as the list of applied policies that led to the action(s). It can also be used to view and respond to any operator notifications issued by the platform as well as operator initiated actions and responses.

4. Contribution

Technology innovations in Broken English parsing, holistic entity disambiguation, adjudication and webpage failure detection and correction are at the heart of MONGOOSE. Our contributions to the field in these areas are detailed in [1, 2, 10, 15]. Another research challenge that was overcome was that of noise effects versus freshness. There is a tension between the desire for rapid and frequent updates reflecting the very cutting edge of what is hot, and minimising the influence of noise in the charts due to short term spikes. Striking a balance here poses an interesting challenge. Effects such as weekends, nights and holidays need to be weighed against events such as new album releases, celebrity gossip events and award shows. Any such system will ultimately be a compromise between being too sensitive and not reactive enough and optimising this balance is a difficult research challenge. To solve this issue, we have used a 24 hour window (that is 4 6-hour cycle periods) to smooth out some effects. Other approaches such as long (multi-month) decays have also been explored. Ultimately, it is important to have a ranking scheme that is at least somewhat resistant to "noise", while still capturing freshness. For example, one that looks for a rise in interest in diverse sources and ignores sudden spikes in a single source.

Another contribution is in the space of spam and on-topic detection for online text documents. The tremendous popularity enjoyed by websites such as MySpace and YouTube also attracts undesirable attention. Spammers and other commercial sites regularly attempt to peddle their content via these sites, by masquerading as "bands" or "users". This poses a challenge that has two distinct flavours. The first one is the ability to distinguish valid artists from those that are nearly product spam. A subject matter topical dictionary enables a first pass at this, as does a list of fairly common "spam" phrases, but the ultimate editorial adjudication at this point is subject matter expert driven. The second is the ability to filter spam and profanities.

On the engineering and impact fronts, MONGOOSE has a compelling value proposition for services practitioners. The advantages of using MONGOOSE to build one's data analytics services offerings are:

1. it speeds up development of new technology in a targeted space, which means faster development due to focus on the core issues rather than focusing on building ingest technology,
2. it is a reusable platform, which reduces the financial and technical challenges in quickly developing and deploying text analytics applications,
3. it detects and responds to failures quickly and the Control Flow is generic across all applications and domains,
4. it handles real-world text, which is messy, has poor syntax and little structure or grammar, and
5. it provides data Integrity and consistency, i.e. clean, consistent data sets over time over the domain of interest.

It is also worth re-emphasizing that MONGOOSE technology is best suited:

1. in domains where Proper Language Analysis (traditional NLP) doesn't work and one needs to apply heuristics on the fly,
2. in Data-Driven Agile Development Environments. MONGOOSE enables agile deployment, which facilitates iterative and incremental development - the data gathered from the sources go in the development process cycle in real-time, and
3. when ingest, pre-processing and rudimentary analysis of dirty data is not the focus of the effort or project.

MONGOOSE serves as an enabler for the new era in business innovation. It demonstrates the next wave in delivering better services and products – the real-time integration of multiple, relevant online information with one's own data to drive new and significant value for, re-invigorate connection to and strengthen brand affinity to one's customer base.

5. Related Work

Most projects that aim to crawl, ingest and process data from unreliable sources such as the Internet end up having to address the challenges described in this paper. The WebFountain architecture [8] included a cluster management system to control and semi-automatically address some common failures in the system. UIMA [7], which primarily focuses on providing a common data mining infrastructure, implements sophisticated logging and error reporting, because the failure of one data miner may affect dependent mining modules. MONGOOSE's contribution is to provide a framework that automates the handling of such error cases.

MONGOOSE is directed towards projects that aim to provide large-scale high availability. Point solutions for storage or computational resources exist, such as Google's Bigtable [4] and MapReduce[5], Amazon's Dynamo [6] and Apache's Hadoop [3]. However, these systems ensure the proper operation of distributed computation and storage, rather than focusing on recovering from failing nodes in these distributed systems. An implicit assumption made by all these systems is that desired data is available, and they simply try to optimize the storage and computation around it. MONGOOSE, in contrast, fetches and cleans the desired data while handling a variety of errors, such that the data can then be processed using the aforementioned systems. It focuses on the end-to-end processing of data, which often requires in-depth knowledge of the processes and expected output of the system.

Taken in the larger context, MONGOOSE is attempting to address a specific instance of the Byzantine fault tolerance problem [14], viewing the total system of data supplier through analytics through publishing of the adjudicated results as a distributed system. Even in the worst case of data supplier or analytics errors, maximally correct results need be generated.

6. Deployments

We have developed solutions based on MONGOOSE for Media and Entertainment [10, 15], Automotive [11] and Healthcare [12, 13]. Although a detailed description of a MONGOOSE deployment is out of the scope of this paper, the SoundIndex implementation is thoroughly presented in [15].

7. Conclusion and Future Work

With the proliferation of Web X.0 data and the rapid rate of services development on this platform, multi-modal content both on the Web and within enterprises, service solutions that attempt to ingest and harness the knowledge embedded therein are up against a significant challenge. Not only are they required to deliver unique insights through sophisticated analytics, but they need to be able to handle multi-modal content ingestion, associated failures, data

integration and pre-processing issues *before* any analytics can be performed.

MONGOOSE enables such information service systems to "get there faster", while minimizing the ongoing pain associated with building and maintaining such systems.

Currently, a quantitative evaluation of the impact of MONGOOSE is difficult; as it has been used to build novel systems where there was no pre-existing solution. On our future work agenda is to create instances where such an evaluation is possible. A naive method would require developing two service systems in parallel, one using MONGOOSE and the other by conventional means, and then measuring the "time to delivery". A relatively straightforward way of determining the reduced maintenance overhead is to convert existing information services to MONGOOSE-enabled services.

In addition to this, we plan to deploy MONGOOSE for other domains to continue building on the domain cartridges. The goal is that over time, there will be sufficient domain-knowledge built into MONGOOSE such that any new system simply needs to select the appropriate MONGOOSE library for their domain. To this end, we also plan to release a suite of MONGOOSE components that can be easily integrated into existing systems.

8. References

[1] Alba, A., Bhagwan, V., and Grandison, T. 2008. Accessing the deep web: when good ideas go bad. In *Companion To the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications* (Nashville, TN, USA, October 19 - 23, 2008). OOPSLA Companion '08. ACM, New York, NY, 815-818.

[2] Alba, A., Bhagwan, V., Grace, J., Gruhl, D., Haas, K., Nagarajan, M., Pieper, J., Robson, C., and Sahoo, N. 2008. Applications of Voting Theory to Information Mashups. In *Proceedings of the 2008 IEEE international Conference on Semantic Computing* (August 04 - 07, 2008). ICSC. IEEE Computer Society, Washington, DC, 10-17.

[3] Apache. Hadoop. http://lucene.apache.org/hadoop/, 2009.

[4] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!). OSDI 2006: 205-218

[5] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150

[6] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. 2007. Dynamo: amazon's highly available key-value store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA, October 14 - 17, 2007). SOSP '07. ACM, New York, NY, 205-220.

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

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

[9] Nagios, http://www.nagios.org

[10] BBC Sound Index project, http://www.almaden.ibm.com/cs/projects/iis/sound/

[11] Zakharian, Z., Mishra, M., Chandramohan, S. "Cars 2.0". Masters Thesis, San Jose State Univeristy, December 2008.

[12] Health-e-Assistant project, http://www.almaden.ibm.com/cs/projects/iis/hea/

[13] AALIM, http://www.almaden.ibm.com/cs/projects/aalim/

[14] Castero, M and Liskov, B. "Practical Byzantine Fault Tolerance". Operating Systems Design and Implementation Feb 1999.

[15] Varun Bhagwan, Tyrone Grandison, Daniel Gruhl, "Sound Index: Music Charts By The People, For The People". To appear in Communications of the ACM. September 2009. Vol 52, No 9.

[16] Yaukey, John. Feds test new data mining program. USA Today. http://www.usatoday.com/news/washington/2007-03-07-datatools_N.htm. March 7, 2007