Architectural Considerations for Scalable Indic Document Analytics

Sai Susarla
NetApp
Bangalore
sai.susarla@gmail.com

Parag Deshmukh
NetApp
Bangalore
parag.d1@gmail.com

K. Gopinath
Computer Science and Automation
Indian Institute of Science Bangalore
gopi@csa.iisc.ernet.in

Abstract

Indic heritage knowledge is embedded in millions of manuscripts at various stages of digitization and analysis. Though numerous powerful tools have been developed for linguistic analysis of Samskrit texts, employing them together on large document collections and building end-user applications is a challenge due to non-standard interfaces. This paper examines the architectural needs of scalable Indic document analytics, and presents our experience in building an actual system. Though it is a work in progress, we demonstrate how careful metadata design enabled us to rapidly develop useful applications via extensive reuse of state-of-the-art analysis tools. This paper offers an approach to standardization of linguistic analysis output, and lays out guidelines for Indic document metadata design and storage.

1 Introduction

There is a lot of interest and activity in applying computing technology to unearth the knowledge content of India's heritage literature especially in Samskrit language. This has led to several research efforts to produce analysis tools for Samskrit language content at various levels – text, syntax, semantics and meaning (Goyal et al., 2012; Kumar, 2012; Huet, 2002; Kulkarni, 2016; Hellwig, 2009). As this field is still in its infancy, these efforts have so far been addressing algorithmic issues in specific linguistic analysis problems. However, as the tools mature and proliferate, it becomes imperative to make them interoperable to solve higher order document analytics problems that involve larger document sets with high performance. For instance, though several alternative linguistic tools exist for Samskrit text analysis (morphological analysis, grammatical checking), they use custom formats to represent input text and analysis outcome, mainly designed for direct human consumption, and not for further machine-processing. This inhibits the use of those tools to build end-user applications for cross-correlating texts, glossary indices, concept search etc.

On the other hand, the number of Heritage Indic documents yet to be explored is staggering. Survey data from National Mission for Manuscripts (NAMAMI, 2012) indicate that there are 3.5 million palm leaf manuscripts in various centers in India. Of them about 2 million have been catalogued, but less than 102,000 texts scanned (having 10 million pages totaling 300TB in size), let alone converted to Unicode text. In addition, The Internet Archive project has a huge collection of scanned printed Indic books. Very few of them has been converted to text. There are also hundreds of thousands of online Unicode Samskrit documents yet to be analyzed grammatically. Use of technology is a must to address this scale.

We believe that to take Indic knowledge exploration to the next level, there needs to be a systematic, end-to-end, interoperability-driven architectural effort to store, exchange, parse, analyze and mine Indic documents at large scale. Due to lack of standardized data representation and machine interfaces for tools, Indic document analysis is unable to leverage numerous advances in data analytics that are already available for English and other languages.

In this paper, we outline the analytics needs of heritage Indic documents, architectural needs to mine large document sets and propose a metadata representation that enables interoperability as a first step.

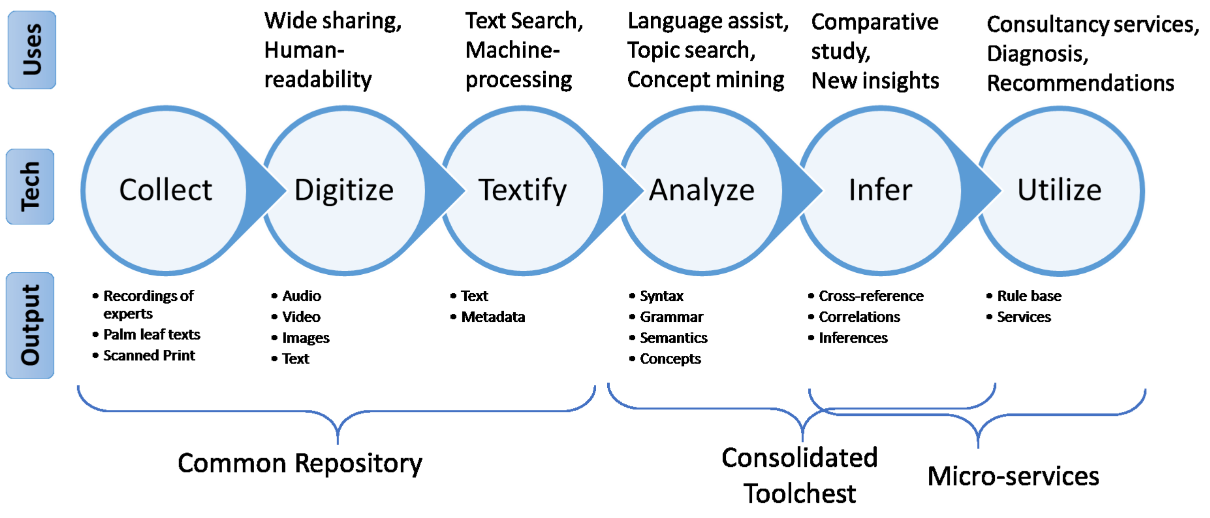
Uses
Tech
Output
Wide sharing, Human-readability
Text Search, Machine-processing
Language assist, Topic search, Concept mining
Comparative study, New insights
Consultancy services, Diagnosis, Recommendations
Collect
Digitize
Textify
Analyze
Infer
Utilize
• Recordings of experts • Palm leaf texts • Scanned Print
• Audio • Video • Images • Text
• Text • Metadata
• Syntax • Grammar • Semantics • Concepts
• Cross-reference • Correlations • Inferences
• Rule base • Services
Common Repository
Consolidated Toolchest
Micro-services

Figure 1: A Workflow for Indic Document Analytics

2 Indic Document Analytics: Overview

Heritage Indic documents come in all media formats and sizes. They include palm leaf and other manuscripts containing hand-written text preserved over millenia, books printed over the last 2 centuries, audio/video recordings of speeches delivered by traditional scholars, and thousands of Unicode texts available over the web. Some of these have been digitized, but not yet converted to machine-processable text. They come in dozens of Indic scripts, languages and fonts in multiple combinations (NAMAMI, 2016), making their organization and processing an engineering challenge. In addition, much of Indic tribal knowledge is still locked up as regional traditions yet to be recorded and captured from their practitioners. Many Indic documents use languages with similar grammatical structure to Samskrit. Samskrit literature is well known to have a rigorous linguistic discipline that makes it more amenable to machine-processing and automated knowledge extraction than other natural languages (Goyal et al., 2012). We use the term Indic Document Analytics (IDA) to denote services to explore the content of Indic documents at various levels – text extraction, syntactic and semantic analysis, knowledge search, mining, representation and inference.

The potential for automated mining of Indic knowledge due to its linguistic base of Samskrit, coupled with the sheer size of Indic document corpus yet to be examined, opens the opportunity to pursue Scalable Indic Document Analytics as an impactful research area in computing. This area is inherently multi-disciplinary, and involves rich media analytics (of audio, video, images), machine-learning, computational linguistics, graph databases, knowledge modeling and scale-out cloud architecture.

Figure 1 illustrates the various stages of a typical IDA workflow covering three distinct transformations: media to text, text to concept, and concept to insight. Each of these stages produces a high volume of metadata in the form of analysis output, content indexes and user feedback that need to be persisted.

2.1 Requirements of an IDA Platform

In addition to scalability and performance to handle millions of documents by thousands of simultaneous users, an IDA platform must have the following properties:

Durability: It must provide both data and metadata persistence, so users can build on prior analysis by others.

Extensibility: The platform must support functional extensions to its services via APIs. It should also provide well-documented data formats and interfaces to incorporate existing and new analytics tools into its fold. This allows existing analysis tools to be reused in larger contexts than intended originally.

Crowd-sourcing: Ambiguity is inherent in natural language understanding. To help resolve ambiguity in analysis and enable users to enrich each other's knowledge through the platform, it must accept human feedback and adapt to it (analogous to Wikipedia). However to reduce user burden, the IDA system must have built-in intelligence to auto-apply suggested corrections to similar contexts.

3 Architectural Considerations for IDA

We now discuss several architectural implications of the above requirements.

3.1 Human-assisted Analytics

Machine processing of Indic documents is still prone to errors and ambiguity, For instance, morphological analysis (Kulkarni, 2016; Huet, 2002) of a Samskrit sentence produces alternative semantic trees sometimes running into hundreds. Text segmentation to detect words from a punctuation-free Indic character sequence can also generate multiple alternative segmentations. Such tools still need human intervention both to supply the context to prune the choices during analysis, and to select a meaningful option from analysis output. Further, system adaptation needs to be built in to create self-improving analyzers. All this requires a mechanism to capture human feedback persistently and incorporate it into analysis. The IDA architecture should provide user-feedback-driven adaptation as a value-addition on top of individual analysis tools, and define standard interfaces to exchange that information with the tools.

3.2 Handling Data Diversity

The input data for an IDA workflow are source documents, which are mostly read-only content. The document analysis tools augment original content with one or more alternate views (e.g., morphological analysis of a sentence, a concept map, an OCR output). When a user annotates those views, some of them become irreplaceable and hence must be stored durably. From a mutability standpoint, an IDA system must deal with three types of content with different rates of churn:

Read-only Source Content that is written once and never updated,

Mutable System-inferred Content that can be reproduced by re-running analytics, and

Mutable Human-supplied Content including user annotations and corrections to system-inferred content.

IDA's data store should clearly demarcate these three types and treat them differently to avoid imbalance in storage performance. Also, for the same source content, there could be multiple alternate views at multiple levels of semantics and granularity that need to be tracked as such. For instance, there could be a sentence-level analysis, paragraph-level analysis and global analysis that coexist for a document.

3.3 Storage Considerations

Rich media documents (audio, video, images, HTML markup text) can be bulky in size, but are read-only. Scalable object stores such as Ceph (RedHat, 2016) are well-suited to such data. Machine-generated analysis output is semi-structured (typically hierarchical key-value markup such as JSON (JSON, 2000) or XML). An example can be found in Algorithm 1. Such data is not only retrieved often for visualization, but also needs to be frequently deleted and recreated as part of analysis workflows. Human annotations require durable persistence, and are not as heavily updated. Fast and scalable semi-structured databases such as NoSQL and graph stores (Neo4j, 2016) are suited for analytics output. Due to their diverse access characteristics, machine-generated and user-supplied content must be segregated in storage for good performance.

3.4 Designing a Canonical Representation for Document Metadata

Standardizing the format of metadata is essential for interoperability of document analysis tools. Recognizing the inherent diversity in Indic document metadata as outlined above is key to designing its right representation for high-performance analytics. In general, a hierarchical key-value representation such as JSON or XML is suitable as output format due to its flexibility and nesting structure. Typical document analysis metadata consists of the following parts:

Source Document Segments: Segment is a portion of the source document to be analyzed, and is identified by its source coordinates. The format of the coordinates varies based on the media type: e.g., Bounding box coordinates in case of a scanned page, time interval in case of an audio/video stream, or text offset and size in case of Unicode text. Document segments can be identified explicitly by users, via automatic segmentation, or via user corrections to auto-generated segments. In case of user-modified segments, their coordinates must be stored persistently, e.g., paragraph bounding boxes in a scanned page. Finally, segments can be nested within other segments. This enables hierarchical analysis.

Annotation: An annotation describes the output of analysis of a source document segment, typically as a set of attribute-value pairs. Annotations must be tagged as system-inferred, user-supplied or user-endorsed. The latter indicates that the user has accepted a system-inferred annotation as valid. Multiple alternate annotations can exist for a given segment, and some of them might be user-endorsed. Annotations can themselves be nested, e.g., a sentence's annotation contains its individual word annotations. Such nesting is indicated by the nesting structure of their referred segments. For re-running analytics on a document segment, a frequent prior operation is to remove earlier system-inferred annotations and all their nested annotations.

Storing segment information and annotations separately enables their independent modification. Algorithms 2 and 1 show two example analysis results from a Grammar RESTful service (VedaVaapi, 2016) we built on top of Samsaadhani toolkit (Kulkarni, 2016). The query in Algorithm 2 requests morphological analysis of the word 'gachChati' in Samskrit but hints that it needs a subanta form. The Samsaadhanii analyzer in the backend returns two subanta and one ti~Nanta forms, but the RESTful service prunes the backend results to two based on requestor's hint. The query in Algorithm 1 requests for a 'loT' lakaara transformation of 'gachChati' as a verb in bahu vachanam. To compute the latter, the transform API implementation invokes the morph analyzer to get the verb root, then the verb-form generator of Samsaadhanii and composes the reply. The ability to take hints and prune the analysis results can thus be used to guide the underlying linguistic tools in the desired manner. These hints can be user-supplied, or inferred from the larger context of a sentence (such as from previous sentences in a passage being analyzed).

4 Implementation

We are prototyping a scalable Indic Document Analytics service called *Vedavaapi* that embodies the key architectural principles outlined earlier. The objective of Vedavaapi is to enable end-user applications that accelerate study of heritage Indic texts. Its approach is to reuse existing linguistic and other analytics tools where possible, apply them to large document collections by bridging gaps in metadata and tools. Figure 2 illustrates the architecture of Vedavaapi service. It employs a micro-services-based architecture with the backend metadata stored in a scale-out MongoDB document database (MongoDB, 2016).

To evaluate the flexibility of this approach, we have built two prototype applications so far:

1. A sentence synthesizer to assist in Samskrit learning. To enable this, we developed a Grammar REST API (VedaVaapi, 2016) that uses the Samsaadhani toolkit for Samskrit Linguistic analysis (Kulkarni, 2016). The API returns its analysis of words and sentences using a JSON-formatted annotation metadata based on the principles outlined in the previous section, and illustrated in Algorithms 1 and 2. Given a Samskrit sentence in Unicode text, the synthesizer uses the API to transform its words into other grammatical forms, suppresses some of the words, quizzes the learner and evaluates their answers.

Algorithm 1 Example Grammar RESTful API query and its results in JSON format. An annotation is an output of such an analysis that refers to the original word.

```
A Grammar RESTful API to transform word 'gachChati':
http://vedavaapi.org/grammar/transform?
    word=gachChati&encoding=Itrans&type=ti~Nanta&
    out_lakara=loT&out_vachana=bahu
and its response:
{
    "result": {
      "dhatu": "gamL^i.N",
      "encoding": "Itrans",
      "gana": "bhvAdiH",
      "lakara": "loT",
      "meaning": "gatau",
      "padi": "parasmai",
      "prayoga": "kartari",
      "purusha": "prathama",
      "result": "gachChantu",
      "root": "gam",
      "vachana": "bahu"
    },
    "status": "ok"
}
```

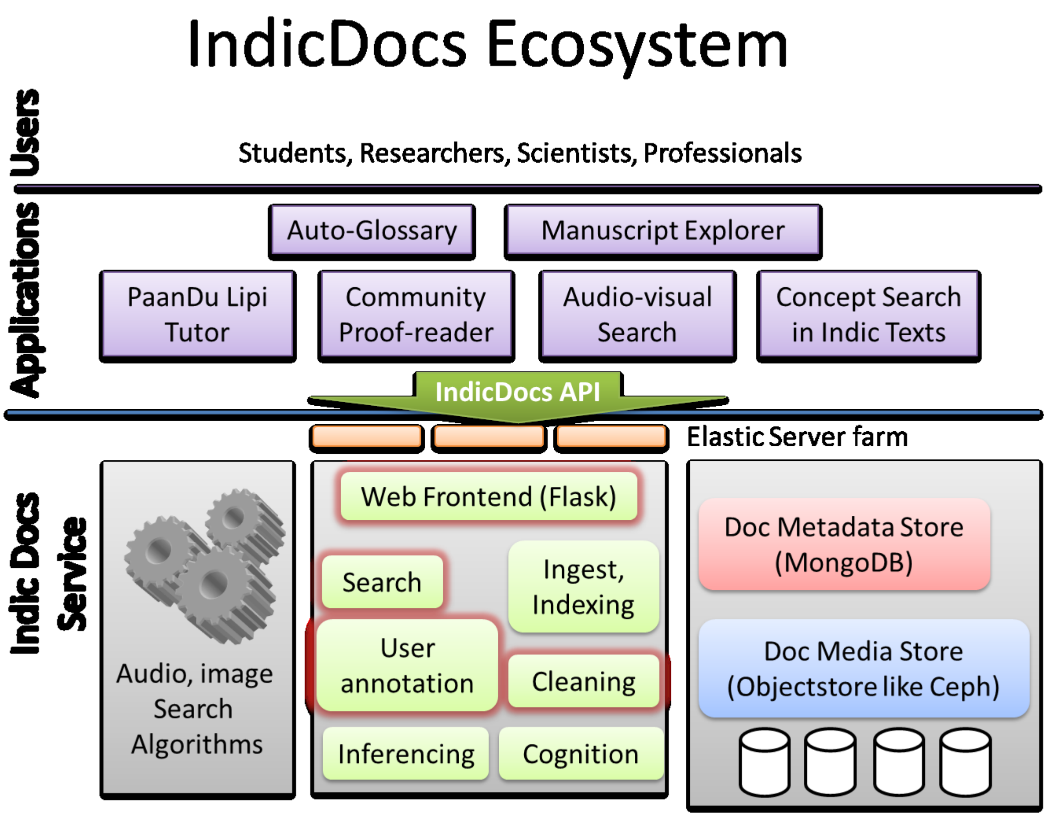

Figure 2: Vedavaapi: An example IDA architecture. The components highlighted in red are currently implemented. IndicDocs is the previous name of Vedavaapi.

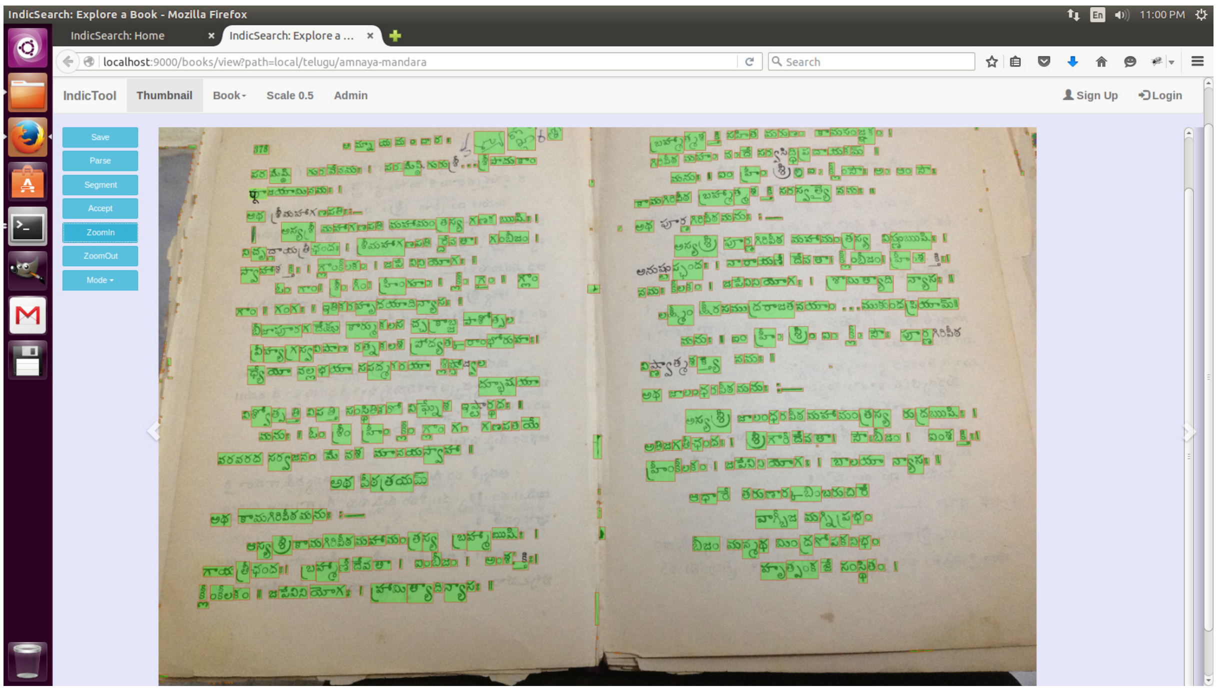
IndicSearch: Explore a Book - Mozilla Firefox
IndicSearch: Home
IndicSearch: Explore a ...
localhost:9000/books/view?path=local/telugu/amnaya-mandara
Search
IndicTool
Thumbnail
Book
Scale 0.5
Admin
Sign Up
Login
Save
Parse
Segment
Accept
ZoomIn
ZoomOut
Mode

Figure 3: Textract: A Visual Text Extraction Application on Vedavaapi platform.

2. A web-based human-assisted text extraction tool (Textract, 2016) for manuscript and scanned printed text images that is agnostic to language, script and font. This tool helps build a crowd-sourced corpus of image-to-text mappings for training existing OCR engines on unknown scripts, fonts and languages. We have employed the document metadata schema for image documents. We have a simple image auto-segmenter that detects the word segments from scanned text. The tool invokes the segmenter and lets user annotate detected segments with typed text. The system then auto-propagates the text to similar-looking character segments in the rest of the book, and lets the user correct any misidentified characters manually. For auto-propagation, we currently employ very simple image template matching, which can be replaced by sophisticated OCR engines. This illustrates a complete end-to-end workflow and the benefits of tools that operate on common document metadata.
   Figure 3 shows a screenshot of this application auto-segmenting a Telugu book photographed using a Smartphone. Each of the green colored rectangles is stored as a segment with its coordinates {x, y, width, height} in the page's image. The tool currently relies on a user to supply what area of the image constitutes an unbroken character sequence such as a paragraph. Once demarcated, those segments can then be annotated with their constituent text. The same segment's text can later be analyzed and annotated at grammatical level, semantic level and conceptual summary level.

5 Future Directions

The metadata architecture laid out in this paper enables several powerful applications including text search within scanned documents, sentence tagging, concept mapping, discourse analysis and auto-glossaries of shaastra texts. Moreover, all of them need human guidance for accurate results. We believe the architectural choices suggested in this paper enable supervised operation to be transparently added to existing analytics tools. We plan to build a semantic concept mapping tool for Indic shaastra texts as an example to validate this hypothesis.

6 Conclusions

In this paper, we have discussed issues in building scalable Indic document analytics services, and presented metadata organization decisions that enable interoperability, rapid development and scaling. We

Algorithm 2 Example Grammar RESTful API query and its results. An annotation is an output of such an analysis that refers to the original word.

```
A Grammar RESTful API to analyze 'gachChati' as a subanta word:
http://vedavaapi.org/grammar/properties?
    word=gachChati&encoding=Itrans&type=subanta
and its response:
{
   "result": {
       "encoding": "Itrans",
       "input": "gachChati",
       "out_encoding": "Itrans",
       "result": [
            {
                "level": 2,
                "linga": "puM",
                "root": "gachChat",
                "subtype": "kR^idanta",
                "type": "subanta",
                "vachana": "eka",
                "vibhakti": 7
            },
            {
                "level": 2,
                "linga": "napuM",
                "root": "gachChat",
                "subtype": "kR^idanta",
                "type": "subanta",
                "vachana": "eka",
                "vibhakti": 7
            }
        ],
        "type": "subanta"
    },
    "status": "ok"
}
```

have approached Indic document analytics from a systems architecture perspective. Though we do not have a computational linguistics background, we feel that these two disciplines can complement each other to enable large-scale exploration of Indic knowledge repositories. The first step is standardization of tool interfaces and data exchange formats. Our initial experience indicates that this is possible and yields rich dividends.

Acknowledgements

Our heartfelt gratitude and thanks to Prof. Amba Kulkarni for providing access to her excellent toolkit to build on, and for encouraging us to write about this work. The foundation for this work was laid two years ago by a generous grant from Samskrit Promotion Foundation to develop an earlier prototype of the textract utility. Our sincere thanks to Sri Ravikiran Sarvadevabhatla, senior PhD candidate of SERC at IISc Bangalore whose insights during brainstorming helped shape the Vedavaapi vision. We deeply appreciate the discussions and numerous application ideas by Sri MVR Sastry of Samskrita Bharati, especially the grammar-based application for Samskrit E-learning.

References

Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In *24th International Conference on Computational Linguistics (COLING), Mumbai.*

Oilver Hellwig. 2009. Extracting dependency trees from sanskrit texts. *Sanskrit Computational Linguistics 3, LNAI 5406*, pages 106–115.

Gérard Huet. 2002. The Zen computational linguistics toolkit: Lexicon structures and morphology computations using a modular functional programming language. In *Tutorial, Language Engineering Conference LEC'2002*.

JSON. 2000. Introducing json. http://www.json.org/.

Amba Kulkarni. 2016. Samsaadhanii: A Sanskrit Computational Toolkit. http://sanskrit.uohyd.ac.in/.

Anil Kumar. 2012. *Automatic Sanskrit Compound Processing*. Ph.D. thesis, University of Hyderabad.

MongoDB. 2016. MongoDB NoSQL Database. http://www.mongodb.com/.

NAMAMI. 2012. Performance Summary of the National Mission for Manuscripts, New Delhi, India. http://namami.org/Performance.htm.

NAMAMI. 2016. National manuscript mission, new delhi, india. http://namami.org/.

Neo4j. 2016. neo4j: The World's leading Graph Database. http://www.neo4j.com/.

RedHat. 2016. Ceph Object Storage. http://www.ceph.com/.

Textract. 2016. VedaVaapi: Text Extraction from Images. http://vedavaapi.org/textract/.

VedaVaapi. 2016. VedaVaapi: Samskrit Grammar Service. http://vedavaapi.org/grammar/.