DISTRIBUTED XML PROCESSING OVER VARIOUS TOPOLOGIES
Pipeline and Parallel Processing Characterization

Yoshiyuki Uratani[1], Hiroshi Koide[2], Dirceu Cavendish[3] and Yuji Oie[3]
[1]*Global Scientific Information and Computing Center, Tokyo Institute of Technology, O-okayama 2-12-1, Meguroku, Tokyo, 152-8550, Japan*
[2]*Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Kawazu 680-4, Iizuka, Fukuoka, 820-8502, Japan*
[3]*Network Design Research Center, Kyushu Institute of Technology, Kawazu 680-4, Iizuka, Fukuoka, 820-8502, Japan*

Keywords: Distributed XML Processing, Task Scheduling, Pipelining and Parallel Processing.

Abstract: This paper characterizes distributed XML processing on networking nodes. XML documents are sent from a client node to a server node through relay nodes, which process the documents before arriving at the server. According as the node topology, the XML documents are processed in a pipelining manner or a parallel fashion. We evaluate distributed XML processing with synthetic and realistic XML documents on real and virtual environments. Characterization of well-formedness and grammar validation processing via pipelining and parallel models reveals inherent advantages of the parallel processing model.

1 INTRODUCTION

XML technology has become ubiquitous on distributed systems, as it supports loosely coupled interfaces between servers implementing Web Services. Large XML data documents, requiring processing at servers, may soon require distributed processing, for scalability. On the other hand, virtualization technology is in rapid development: virtualization brings benefits such as efficient administration, economic electricity consumption, small running cost, and robustness. As current Web Services run on virtual machines, it is valuable to study distributed XML processing on virtual environments, in addition to real environments.

Recently, distributed XML processing has been proposed and studied from an algorithmic point of view for well-formedness, grammar validation, and filtering (Cavendish and Candan, 2008). In that work, a Prefix Automata SyStem (PASS) is described, where PASS nodes opportunistically process fragments of an XML document travelling from a client to a server, as a data stream. PASS nodes can be arranged into two basic distributed processing models: pipelining, and parallel models. We have also studied task allocation of XML documents over pipeline and parallel distributed models in (Uratani et al., 2011).

In this paper, we augment processing characterization scope by introducing realistic XML documents, as well as characterization of XML distributed processing on virtual machines, in addition to real machines. We provide a comprehensive characterization of distributed XML processing by evaluating two distributed XML processing models - i) pipelining, for XML data stream processing systems; ii) parallel, for XML parallel processing systems. We conduct such evaluation on real and virtual environments using two types of XML documents; synthetic and realistic documents. Our results can be summarized as follows. Parallel processing performs better than pipeline processing; there are little differences between real and virtual environments; document characteristics (e.g. tags count) and processing environments impact performance of distributed XML processing.

The paper is organized as follows. In section 2, we describe generic models of XML processing nodes. In section 3, we describe various experimental environments that implement the distributed XML processing system, and characterize XML processing performance of our experiments. In section 4, we address related work, then we summarize our findings and address research directions in section 5.

Uratani Y., Koide H., Cavendish D. and Oie Y..
DISTRIBUTED XML PROCESSING OVER VARIOUS TOPOLOGIES - Pipeline and Parallel Processing Characterization.
DOI: 10.5220/0003933601160122
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 116-122
ISBN: 978-989-8565-08-2

2 XML PROCESSING ELEMENTS

Distributed XML processing requires some basic functions to be supported: **Document Partition:** The XML document is divided into fragments, to be processed at processing nodes. **Document Annotation:** Each document fragment is annotated with current processing status upon leaving a processing node. **Document Merging:** Document fragments are merged so as to preserve the original document structure. For supporting these tasks, we implemented following four types of nodes. The distributed XML processing can then be constructed by connecting these nodes in specific topologies, such as pipelining and parallel topologies.

StartNode is a source node that executes any preprocessing needed in preparation for piecewise processing of the XML document. This node reads the document from its local storage, adds some annotation (task allocating information and tag checking information) which are used for distributed XML processing, and sends the fragments to a next node. The tag checking information is for processing status indication: already matched; unmatched; or yet to be processed. The StartNode has multiple threads to execute these activity.

RelayNode executes XML processing on parts of an XML document and is placed as an intermediate node in paths between the StartNode and the EndNode. This node has three threads for receiving/processing/sending fragments of XML data, so it can process while receiving/sending the data. These threads share a buffer among each other for collaboration. The RelayNodes check allocating information first, and process document fragments assigned to them.

EndNode is a destination node, where XML documents must reach, and have their XML processing finished. This node receives the XML document and its annotations for processing from a previous node. If the tag checking has not been finished yet the EndNode processes all unchecked tags, in order to complete XML processing of the entire document.

MergeNode receives data from multiple previous nodes, serializes it, and sends it to a next node, without performing any XML processing. For smooth data transfer for each previous node, this node has multiple threads for receiving data.

XML document processing involves stack data structures for tag processing. When a node reads a start tag, it pushes the tag name into a stack. When a node reads an end tag, it pops a top element from the stack, and compares the end tag name with the popped tag name. If both tag names are the same, the tags match. The XML document is well-formed when all tags match. In case the pushed and popped tags do not match, the XML document is declared ill formed. In addition in validation checking, each node executing grammar validation reads DTD files, and generates grammar rules for validation checking. Then each node processes validation and well-formedness at the same time, with comparing the popped/pushed tags and the grammar rules.

3 DISTRIBUTED XML CHARACTERIZATION

In this section, we characterize distributed XML well-formedness and grammar validation processing.

3.1 Experimental Environment

We use two types of environments: real and virtual environments.

PC_Env consists of two types of PC cluster. Their specification is described in Table 1. One of them has 2 CPU cores (It is only assigned to node06); the other has 6 PC clusters each with 4 CPU cores.

VM_Env is based on PC_Env, and consists of a VMware ESX 4 on a Sun Fire X4640 Server. We use VMware ESX 4, a virtual machine manager, to implement virtual machines for using distributed XML processing nodes. The server specification is described in Table 2. We allocate two cores to node06, and four cores to all other nodes, similar to PC_Env.

Table 1: PC Cluster Specification.

	PC 4core	PC 2core
CPU	Intel Core 2 Quad Q965 (3GHz)	Intel Core 2 Duo E8400(3GHz)
Memory	4G Byte	
NIC	1000 BASE-T Intel 8254PI	1000 BASE-T Intel 82571 4 port × 2
OS	Fedora13_x86_64	
JVM	Java™ 1.5.0_22	

Table 2: X4640 Server Specification.

CPU	Six-Core AMD Opteron Processor 8435 (2.6GHz) × 8
Memory	256G bytes (DDR2/667 ECC registered DIMM)
VMM	VMware ESX 4
Guest OS	Fedora15_x86_64
JVM	Java™ 1.5.0_22

Table 3: XML Document Characteristics.

	doc01	doc02	doc03	doc04	doc05	doc06	doc07	kernel	stock	scala
Width	10000	5000	2500	100	4	2	1	-	-	-
Depth	1	2	4	100	2500	5000	10000	-	-	-
Tag set count (Empty tag count)	10000(0)							2255 (36708)	66717 (146)	26738 (1206)
Line count	10002	15002	17502	19902	19998	20000	20001	41219	78010	72014
File size [Kbytes]	342	347	342	343	342			3891	2389	2959

3.2 Node Allocation Patterns

We use several topologies and task allocation patterns for characterizing distributed XML processing, within the parallel and pipelining models with varying the number of RelayNodes and its topology. Figure 1 shows 2 RelayNode pipeline topology, whereas Figure 2 shows 2 RelayNode parallel patterns. In the figures, tasks are shown as light shaded boxes, underneath nodes allocated to process them. For example in Figure 1, we divide the XML documents into three parts: first two lines (it contains meta tag and root tag), fragment01 and fragment02. Data flows from StartNode to EndNode via two RelayNodes. RelayNode02 is allocated for processing fragment02, RelayNode03 is allocated for processing fragment01, and the EndNode is allocated for processing the first two lines, as well as processing all left out unchecked data. In parallel topology, the fragments flow from StartNode to EndNode via each RelayNode and MergeNode, and they are concurrently sent from the StartNode to the RelayNodes. These document partition and allocation patterns are defined beforehand as static task scheduling. When a pair of open and close tags is parted into different fragments, the pair can not be processed at a single RelayNode. We also experiment four RelayNode topology for both processing types with the five parted XML documents.

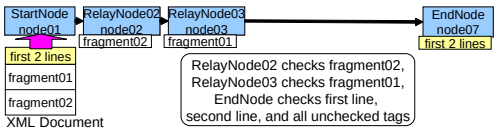

Figure 1: Two Stage Pipeline System.

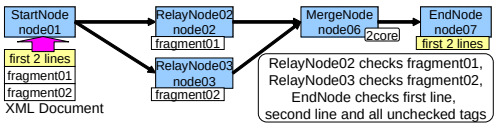

Figure 2: Two Path Parallel System.

3.3 Tasks and XML Document Types

We use different structures of XML documents to investigate which distributed processing model yields the most efficient distributed XML processing. For that purpose, we create seven types of synthetic XML documents by changing the XML document depth from shallow to deep while keeping its size almost the same. We have also used three types of realistic XML documents, doc_kernel, doc_stock and doc_scala. The doc_kernel encodes directory and file hierarchy structure of linux kernel 2.6.39.3 in XML format. The doc_stock is XML formatted data from MySQL data base, containing a total of 10000 entries of dummy stock price data. The doc_scala is based on “The Scala Language Specification Version 2.8” (http://www.scala-lang.org/), which consists of 191 pages, totalling 1.3M bytes. The original document in pdf format was converted to Libre Office odt format, and from that to XML format. Synthetic and realistic XML document characteristics are shown in Table 3.

3.4 Performance Indicators

We use two types performance indicators: system performance indicators and node performance indicators. System performance indicators characterize the processing of a given XML document. Node performance indicators characterize XML processing at a given processing node. The following performance indicators are used to characterize distributed XML processing:

Job Execution Time is a system performance indicator that captures the time taken by an instance of XML document to get processed by the distributed XML system in its entirety. As several nodes are involved in the processing, the job execution time results to be the period of time between the last node (typically EndNode) finishes its processing, and the first node (typically StartNode) starts its processing.

Node Thread Working Time is a node performance indicator that captures the amount of time each thread of a node performs work. It does not include thread waiting time when blocked, such as data re-

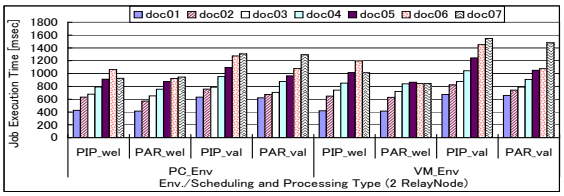

Figure 3: Job Execution Time (Synthetic Docs: 2RN).

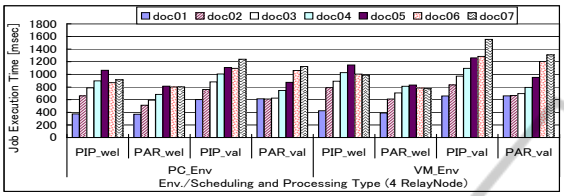

Figure 4: Job Execution Time (Synthetic Docs: 4RN).

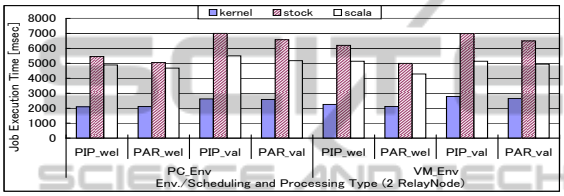

Figure 5: Job Execution Time (Realistic Docs: 2RN).

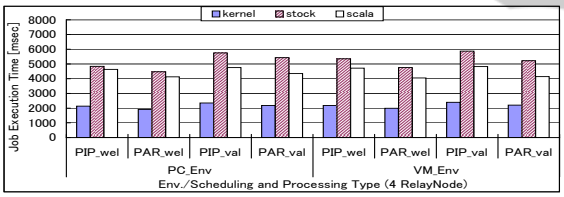

Figure 6: Job Execution Time (Realistic Docs: 4RN).

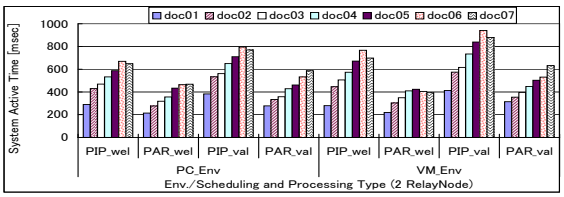

Figure 7: System Active Time (Synthetic Docs: 2RN).

ceiving wait time. It is defined as the total file reading time, data receiving time and data sending time a node incurs. We also derive a **System Thread Working Time** as a system performance indicator, as the average of node thread working time indicators across all nodes of the system.

Node Active Time is a node performance indicator that captures the amount of time each node runs. The node active time is defined from a node starts receiving/reading first data until the node finishes sending last data in the node or finishes document processing in the EndNode. Hence, the node active time may

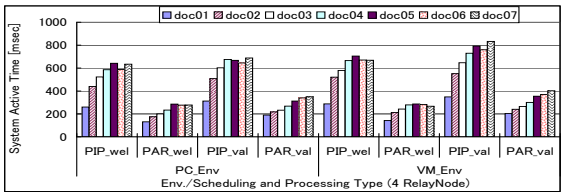

Figure 8: System Active Time (Synthetic Docs: 4RN).

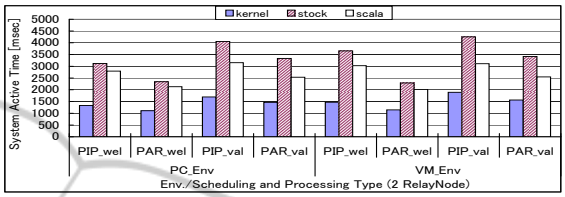

Figure 9: System Active Time (Realistic Docs: 2RN).

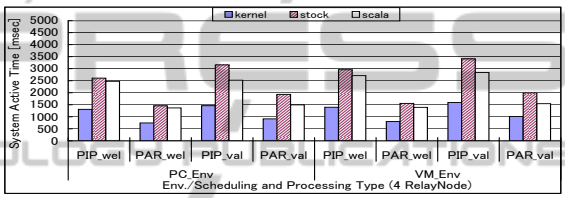

Figure 10: System Active Time (Realistic Docs: 4RN).

contain waiting time (e.g, wait time for data receiving, thread blocking time). We also define **System Active Time** as a system performance indicator, by averaging the node active time of all nodes across the system.

Node Processing Time is a node performance indicator that captures the time taken by a node to execute XML processing only, excluding communication and processing overheads. We also define **System Processing Time** as a system performance indicator, by averaging node processing time across all nodes of the system.

Parallelism Efficiency Ratio is a system performance indicator defined as "*system thread working time* / *system active time*".

3.5 Experimental Results

For each experiment type (scheduling allocation and distributed processing model), we collect performance indicators data over seven types of XML document instances. Figures 3–6 report job execution time; Figures 7–10 report system active time; Figures 11–14 report system processing time; Figures 15–18 report system parallelism efficiency ratio. On all graphs, X axis describes scheduling, processing models and processing environment, for well-formedness and grammar validation types of XML document processing, encoded as follows: **`PIP_wel`**:Pipeline

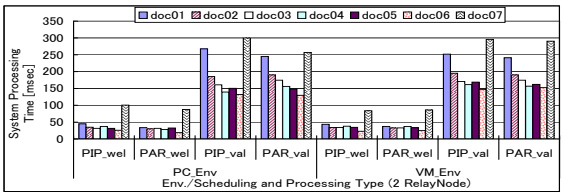

Figure 11: System Processing Time (Synthetic Docs: 2RN).

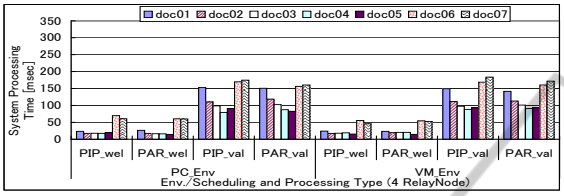

Figure 12: System Processing Time (Synthetic Docs: 4RN).

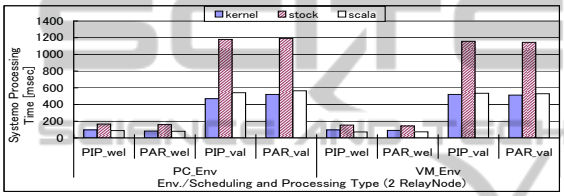

Figure 13: System Processing Time (Realistic Docs: 2RN).

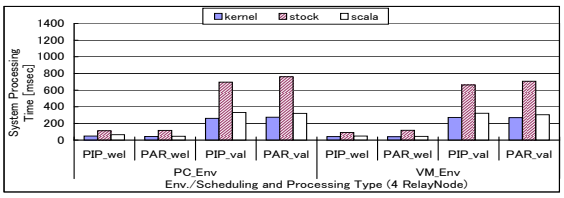

Figure 14: System Processing Time (Realistic Docs: 4RN).

and Well-formedness checking, **PAR_wel**:Parallel and Well-formedness checking, **PIP_val**:Pipeline and Validation checking, **PAR_val**:Parallel and Validation checking. Y axis denotes specific performance indicator, averaged over 22 XML document instances.

In all indicators, both PC_Env and VM_Env show similar characteristics. They are similar qualitatively but different quantitatively, due to differences in document processing. Therefore, we can conclude differences between real and virtual environments are small.

Regarding job execution time, parallel processing is faster than pipeline processing for all documents. Job execution time gets lengthened in pipeline processing due to the fact that the nodes relay extra data that is not processed locally. In addition, we can see that the job execution time speeds up faster with increasing number of relay nodes in parallel processing than in pipeline processing. The extra data transfer

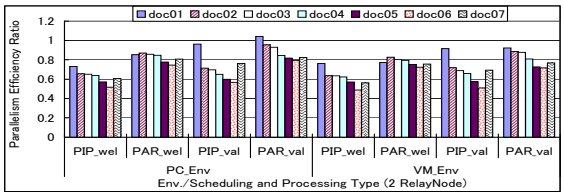

Figure 15: Parallelism Efficiency Ratio (Synthetic Docs: 2RN).

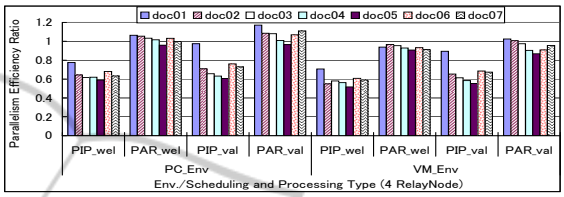

Figure 16: Parallelism Efficiency Ratio (Synthetic Docs: 4RN).

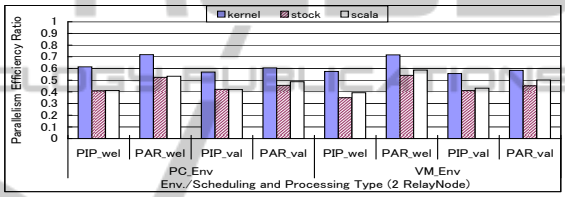

Figure 17: Parallelism Efficiency Ratio (Realistic Docs: 2RN).

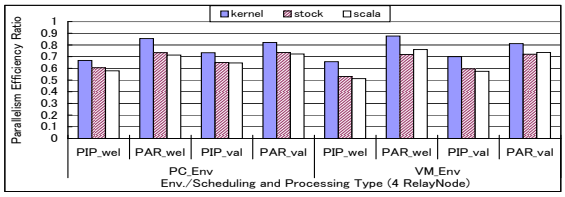

Figure 18: Parallelism Efficiency Ratio (Realistic Docs: 4RN).

time also appears in pipeline processing. Moreover, increased number of relay nodes reduces further job execution time of realistic documents (bigger documents), as compared with that of synthetic documents (smaller documents) processed with pipeline model. So, it is more advantageous to process bigger XML document using the pipeline model with more nodes. Notice that doc_stock is smaller than doc_kernel, and has more XML tags than the doc_kernel. As the doc_kernel has smaller job execution time than the doc_stock, we can say that the job execution time is more sensitive to the number of tags of a document than its size. This characteristic also appears in some other indicators: system active time and system processing time.

Regarding system active time, parallel process-

ing is better than pipeline processing in all experiments. The system active time decreases with increasing number of relay nodes in both synthetic and realistic document processing. In addition, the higher the document depth and the more number of XML tags, the larger the system active time in both documents. Useless processing is more pronounced at each RelayNode for the synthetic documents with higher depth, because the RelayNodes are not able to match too many tags within the document part allocated to them. Hence, the EndNode is left with a large amount of processing left to be done. We may reduce useless processing if we divide the document conveniently according to the document structure and grammar checking rules. In addition, node activity is more sensitive to the number of RelayNodes in parallel processing than in pipeline processing. Regarding task complexity, node activity results in pipeline processing are similar, regardless of the task performed.

Processing time is also similar for both parallel processing and pipeline processing. The results show the extra activity time in the pipeline processing is due to extra sending/receiving thread times. Generally, the system processing time reduces as the number of RelayNodes increases. However, sometimes (e.g. synthetic doc06) few RelayNodes are more efficient, due to specific document partition and processing allocation.

Regarding parallelism efficiency ratio, parallel processing is more efficient than pipeline in every case, because in parallel case, more threads are operating concurrently on different parts of the document. According to the parallelism efficiency ratio, synthetic documents are more efficient than realistic documents for processing. Moreover, in synthetic document processing, the efficiency ratio is better in 4 RelayNode than in 2 RelayNode for parallel processing only. In realistic document processing, the efficiency ratio is improved in 4 RelayNode as compared with 2 RelayNode for both pipeline and parallel processing.

For convenience, we organize our performance characterization results into Table 4.

Table 4: Summary of Experimental Results.

	Synthetic docs (smaller docs)		Realistic docs (larger docs)	
	PAR vs PIP	number of RN	PAR vs PIP	number of RN
Job execution time	PAR is better	4 RN is better in PAR	PAR is better	4 RN is better in both PIP and PAR
System active time				
System processing time	No difference	4 RN is better	No difference	4 RN is better
Parallelism efficiency ratio	PAR is better	4 RN is better in PAR	PAR is better	

4 RELATED WORK

XML parallel processing has been recently addressed in several papers. (Lu and Gannon, 2007) proposes a multi-threaded XML parallel processing model where threads steal work from each other, in order to support load balancing among threads. They exemplify their approach in a parallelized XML serializer. (Head and Govindaraju, 2007) focuses on parallel XML parsing, evaluating multi-threaded parsers performance versus thread communication overhead. (Head and Govindaraju, 2009) introduces a parallel processing library to support parallel processing of large XML data sets. They explore speculative execution parallelism, by decomposing Deterministic Finite Automata (DFA) processing into DFA plus Non-Deterministic Finite Automata (NFA) processing on symmetric multi-processing systems. To our knowledge, our work is the first to evaluate and compare parallel against pipelining XML distributed processing.

5 CONCLUSIONS

In this paper, we have studied two models of distributed XML document processing, parallel and pipeline, on two types of environments, virtual and real machines, using XML documents with various characteristics. In general, both virtual and real environments are similar in streaming data processing of XML documents. Virtual machines are flexible in resource allocation, providing more efficient resource utilization. Regarding processing models, pipeline processing is less efficient than parallel processing in both document type. because parts of the document that are not to be processed at a specific node needs to be received and relayed to other nodes, consuming node resources and increasing processing overhead. Regardless the distributed model, efficiency of distributed processing depends on the XML document characteristics, as well as its task partition. Optimal partition of XML document for efficient distributed processing is part of ongoing research.

In this work, we have focused on distributed well-formedness and validation of single XML documents. Other XML processing, such as filtering and XML transformations, as well as multiple XML document processing, can be studied. We are also interested in

the impact of increasing number of CPU cores of each node on the performance of distributed XML processing, as well as the processing of streaming data other than XML documents at relay nodes (Shimamura et al., 2010). In such scenario, many web servers, mobile devices, network appliances, are connected with each other via an intelligent network, which executes streaming data processing on behalf of connected devices. The type of node processing is different than XML processing, given the less structured nature of streaming data, as compared with XML data.

ACKNOWLEDGEMENTS

Part of this study was supported by a Grant-in-Aid for Scientific Research (KAKENHI:21500039).

REFERENCES

Cavendish, D. and Candan, K. S. (2008). Distributed XML processing: Theory and applications. *Journal of Parallel and Distributed Computing*, 68(8):1054–1069.

Head, M. R. and Govindaraju, M. (2007). Approaching a Parallelized XML Parser Optimized for Multi-Core Processors. *SOCP'07*, pages 17–22.

Head, M. R. and Govindaraju, M. (2009). Performance Enhancement with Speculative Execution Based Parallelism for Processing Large-scale XML-based Application Data. *HPDC'09*, pages 21–29.

Lu, W. and Gannon, D. (2007). Parallel XML Processing by Work Stealing. *SOCP'07*, pages 31–37.

Shimamura, M., Ikenaga, T., and Tsuru, M. (2010). Advanced relay nodes for adaptive network services - concept and prototype experiment. *Broadband, Wireless Computing, Communication and Applications, International Conference on*, 0:701–707.

Uratani, Y., Koide, H., Cavendish, D., and Oie, Y. (2011). Characterizing Distributed XML Processing – Moving XML Processing from Servers to Networking Nodes. *Proc. 7th International Conference on Web Information Systems and Technologies*, pages 41–50.