Distributed XML Processing over Multicore Servers

Yoshiyuki Uratani[1], Hiroshi Koide[2] and Dirceu Cavendish[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, Multicore CPU.

Abstract: Nowadays, multicore CPU become popular technology to enhance services quality in Web services. This paper characterizes parallel distributed XML processing which can off-load the amount of processing at their servers to networking nodes with varying number of CPU cores. Our implemented distributed XML processing system sends XML documents from a sender node to a server node through relay nodes, which process the documents before arriving at the server. When the relay nodes are connected in tandem, the XML documents are processed in a pipelining manner. When the relay nodes are connected in parallel, the XML documents are processed in a parallel fashion. For well-formedness and grammar validation tasks, the parallel processing reveals inherent advantages compared with pipeline processing regardless of document type, number of CPU cores and processing environment. Moreover, the number of CPU cores impacts efficiency of distributed XML processing via buffer access contention.

1 INTRODUCTION

Web services become necessary infrastructure for our society. Various services are being provided and they need much resources, such as CPU power and memory spaces, to enhance their services quality. Nowadays, multicore CPU is popular approach to improve processing capacity. The processing for Web services are generally provided at only server nodes in the current Web services. In this situation, we propose offloading approach, which assign a part of server's load to intermediate nodes in network, to reduce the load and to improve services throughput. This approach will lead efficient resource usage with effective scheduling and higher quality of services. On the other hand, XML data is one of the basic communication format in the Web services and the servers often processes the XML data in various situation (e.g. collaborative services). PASS-Node (Cavendish and Candan, 2008) had already proposed distributed XML processing with using intermediate nodes and had studied from an algorithmic point of view for well-formedness, grammar validation, and filtering. We have also focuses on distributed XML processing with offloading approach and have studied their processing characteristics (Uratani et al., 2012).

This paper augments practical distributed XML processing characterization scope by varying the number of CPU cores in processing nodes; using synthetic and realistic XML documents; on pipelining model for XML data stream processing systems, on parallel model for parallel processing system. Regarding processing efficiency, we investigate XML processing performance relation to XML document characteristics,as well as the impact of number of CPU cores to well-formedness and grammar validation tasks. Our results can be summarized as follows. Parallel processing performs better than pipeline processing regardless the number of CPU cores, document types and processing environment; Processing with more CPU cores leads to faster processing; but it also includes drawbacks, such as inefficient resource consumption, due to buffer contention.

The paper is organized as follows. In section 2, we describe generic models of XML processing nodes. In section 3, we describe experimental environments and characterize XML processing performance of the pipeline and parallel computation models on multicore machines. In section 4, we address related work. In section 5, we summarize our findings and address research directions.

Uratani Y., Koide H. and Cavendish D..
Distributed XML Processing over Multicore Servers.
DOI: 10.5220/0004947802000207
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 200-207
ISBN: 978-989-758-023-9

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. XML processing nodes support some of these tasks, according to their role in the distributed XML system.

2.1 XML Processing Nodes

We abstract the distributed XML processing elements into four types of nodes: StartNode, RelayNode, EndNode, and MergeNode. The distributed XML processing can then be constructed by connecting these nodes in specific topologies, such as pipelining and parallel topologies. Distributed processing applications may be constructed with both parallel and pipeline manner. So the study of pipelined approach is to opportunistically process XML documents on arbitrary network topologies, in the middle of the network.

`StartNode(SN)` is a source node that executes any pre-processing needed in preparation for piecewise processing of the XML document. This node also starts XML document transfer to relay nodes and adds some annotation which are used for distributed XML processing. The StartNode has multiple threads for reading part of the document and sending fragments simultaneously.

`RelayNode(RN)` executes XML processing on parts of an XML document. It is placed as an intermediate node in paths between the StartNode and the EndNode. The RelayNode has three types of threads: ReceiveThread, TagCheckThread and SendThread. The ReceiveThread receives data containing lines of an XML document, together with checking and processing information, and stores the data into a shared buffer. The TagCheckThread attempts to process the data, if the data is assigned to be processed at the node. SendThread sequentially sends data to a next node.

EndNode(EN) is a destination node, where XML documents must reach, and have their XML processing finished. This node receives data containing the XML document, checking information and processing information, 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. Components of the EndNode are similar to the RelayNode, except that the EndNode has DeleteThread instead of SendThread. The DeleteThread cleans the document from processing and checking information.

`MergeNode(MN)` receives data from multiple previous nodes, serializes it, and sends it to a next node, without performing any XML processing. MergeNode has multiple threads for receiving data from previous node, so as not to block previous nodes from sending 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 addition, in validation checking, each node executing grammar validation reads DTD files, and generates grammar rules for validation checking. Each node processes validation and well-formedness at the same time, comparing the popped/pushed tags against grammar rules. Details of these node distributed processing is described in (Cavendish and Candan, 2008; Uratani et al., 2012).

3 DISTRIBUTED XML CHARACTERIZATION

3.1 Experimental Environment

We use a **`VM_Env`**, which consists of a VMware ESX 4 on a Sun Fire X4640 Server, for providing distributed XML processing system. We use VMware ESX 4, a virtual machine manager, to implement a total of seven counts of virtual machines as distributed XML processing nodes. This environment consists of two types of virtual machine. We allocate two CPU cores to one of them (Node06), and four CPU cores to all

Table 1: 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

other nodes. The server specification is described in Table 1.

3.2 Node Allocation Patterns

We prepare several topologies and task allocation patterns to characterize distributed XML processing, within the parallel and pipelining models. We vary the number of RelayNodes, within topologies, as well as the number of CPU cores in processing nodes, to evaluate their impact on processing efficiency. To characterize the performance, we use a total of four topology types: two stage pipeline system (Figure 1), two path parallel system (Figure 2), four stage pipeline system, and four path parallel system. In the Figure 1 and 2, tasks are shown as light shaded boxes, underneath nodes allocated to process them. CPU core count is shown above the task boxes.

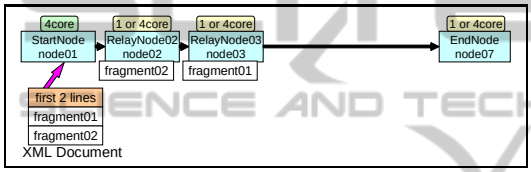

Figure 1: Two Stage Pipeline System.

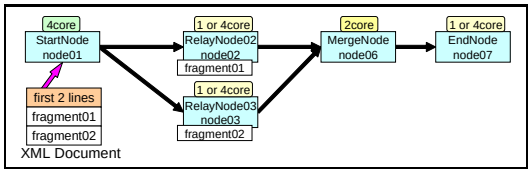

Figure 2: Two Path Parallel System.

For instance in two stage pipeline case (Figure 1), we divide the XML documents into three parts: first two lines (it contain a meta tag and a root tag), fragment01 and fragment02. These fragments are divided into segments of roughly the same size. 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 Figure 2, we depict two path parallel system. The XML document is also divided into three parts. They flow from StartNode to EndNode via RelayNode01 and MergeNode. The StartNode reads concurrently these fragments from the XML document and sends them to the RelayNodes. Fragment02 and related data flow from the StartNode to the EndNode via RelayNode02 and the MergeNode. In addition, we can use a maximum of 4 CPU cores in node02, node03 and node07, which are allocated for the RelayNodes and EndNode. We apply two types of CPU usage patterns in this topology; i) Allocation of 1 CPU core only to RelayNodes and EndNode; ii) Allocation of all 4 CPU cores to RelayNodes and EndNode. These document partition and allocation patterns are defined beforehand as static task scheduling. We also experiment four RelayNode topology for both processing types with the five parted XML documents. In the processing types, we also vary number of CPU cores of the RelayNodes and the EndNode.

Notice that, even though the parallel model has one extra node, the MergeNode, as compared with corresponding pipeline model. However, the MergeNode does not perform any XML processing, so the number of nodes executing XML processing is still the same in both models.

3.3 Tasks and XML Document Types

The distributed XML processing system can execute two types of processing: well-formedness checking, and grammar validation checking of XML documents. Efficiency of these XML processing tasks may be related to: processing model, pipelining and parallel; topology, number of processing nodes and their connectivity; XML document characteristics. 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, totaling 1.3M bytes. The original document in pdf format was converted to Libre Office odt format, and from that to XML format. Their characteristics are shown in Table 2, For VM_Env, we combine four node allocation patterns, two processing patterns and ten XML document types to produce a total 80 types of experiments.

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 perfor-

Table 2: 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 tags)	10000 (0)							225 (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

mance 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 the StartNode) starts its processing. The job execution time is measured for each XML document type and processing model.

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 receiving wait time. It is defined as the total file reading time, data receiving time, data sending time and node buffer access time a node incurs. For instance, in the MergeNode, the node thread working time is the sum of receiving time and sending time of each Receive/SendThread. Assume the MergeNode is connected to two previous nodes, with two Receive/SendThreads. 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 the first ReceiveThread starts receiving first data until the last SendThread finishes sending last data in the RelayNode or finishes document processing in the EndNode. Hence, the node active time may 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*".

Node Buffer Access Time is a node performance indicator that captures the amount of time a node spends accessing internal shared buffers. As previously mentioned in section 2, threads in each node access the shared buffer while receiving/processing/sending data. The node buffer access time includes not only the time for add/get/remove operation, but also waiting time during blocking. We also define **System Buffer Access Time** as a system performance indicator, by totaling the node buffer access time of all nodes across the system.

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. 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 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. These figures are only part of the experimental results - details of other results are omitted for space's sake.

Regarding job execution time (Figures 3 – 6), 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

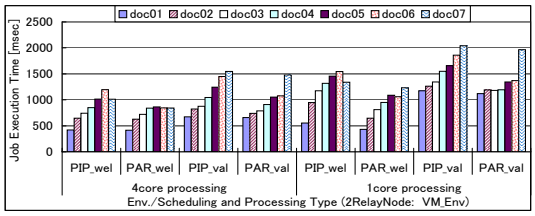

Figure 3: Job Execution Time (Syn. Doc. Proc. by 2 RN).

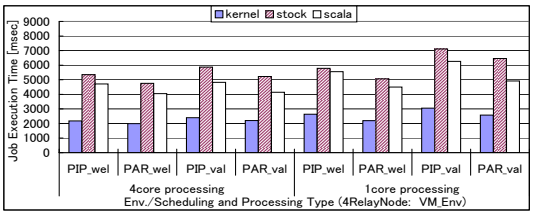

Figure 6: Job Execution Time (Real. Doc. Proc. by 4 RN).

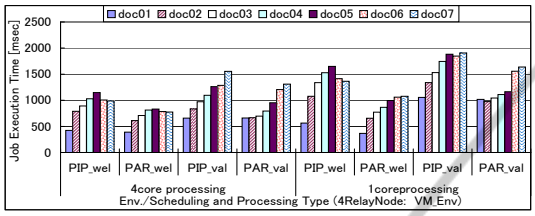

Figure 4: Job Execution Time (Syn. Doc Proc. by 4 RN).

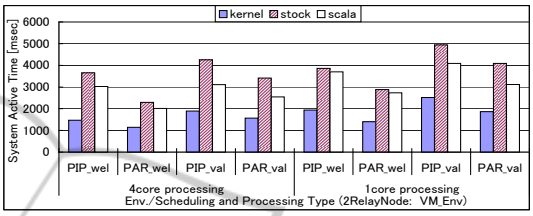

Figure 7: System Active Time (Real. Doc. Proc. by 2RN).

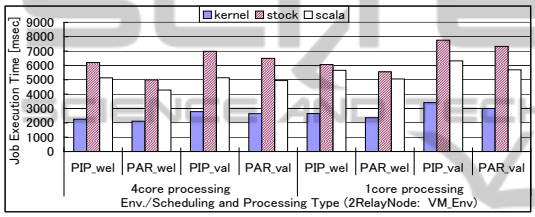

Figure 5: Job Execution Time (Real. Doc. Proc. by 2 RN).

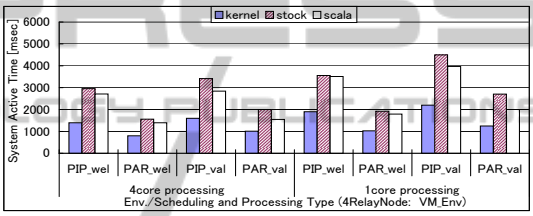

Figure 8: System Active Time (Real. Doc. Proc. by 4RN).

parallel processing than in pipeline processing. The extra data transfer 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. Regarding the number of CPU cores, 4 core processing is better than 1 core processing for both small (synthetic) and large (realistic) document processing. In Table 2, stock_doc is the smallest document in the three types of realistic document, and it has more XML tags than other documents. kernel_doc has more tags than scala_doc but most of them are empty tags. The empty tag processing needs less time than normal tag sets because the nodes can determine results of processing to empty tags earlier. Job execution time is sensitive to document types (number of tags and types of tags), node topology and processing type. Such characteristics also appears in other system indicators.

Regarding system active time (Figure 7 and 8), parallel processing is better than pipeline processing in all experiments. Regarding number of CPU cores, system active time is better for 4 core processing than for 1 core processing in both synthetic documents and realistic documents, regardless of document partition, processing allocation, node topology and buffer contention. The system active time also depends on the amount of processing to be executed.

Processing time is similar in both parallel processing and pipeline processing (Figures 9 and 10). Validation checking needs more time than well-formedness checking. The extra activity time in the pipeline processing is due to extra sending/receiving thread times. Regarding number of CPU cores, 1 core processing is better than 4 core processing in well-formedness checking (small processing). In contrast, 4 core processing is better than 1 core processing in validation checking (large processing) for synthetic documents (small document). But for some realistic documents (bigger document), 1 core processing has better validation checking processing time. If the number of tags that each node should process is roughly the same, 4 core processing leads to more buffer contention than 1 core processing. 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. Average processing time is greatly affected by whether we can allocate XML data efficiently. In addition, system processing time is also sensitive to

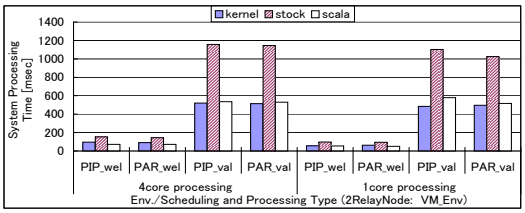

Figure 9: Sys. Processing Time (Real. Doc. Proc. by 2RN).

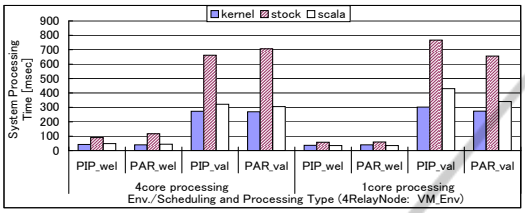

Figure 10: Sys. Processing Time (Real. Doc. Proc. by 4RN).

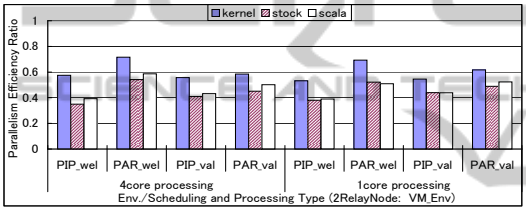

Figure 11: Parallel. Efficien. Ratio (Real. Doc. Proc. by 2RN).

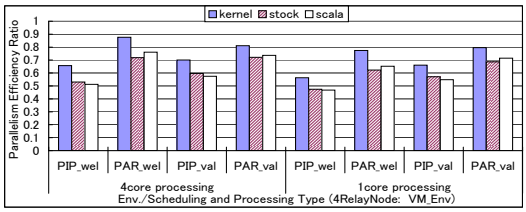

Figure 12: Parallel. Efficien. Ratio (Real. Doc. Proc. by 4RN).

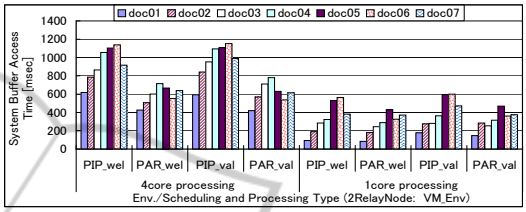

Figure 13: Sys. Buf. Access Time (Syn. Doc. Proc. by 2RN).

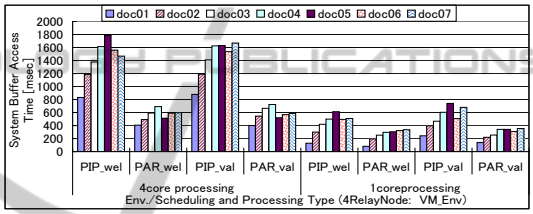

Figure 14: Sys. Buf. Access Time (Syn. Doc. Proc. by 4RN).

XML document structure, number of tags and depth, which affects the amount of processing at each node.

Regarding parallelism efficiency ratio (Figures 11 and 12), 4 core processing is more efficient than 1 core processing excluding few cases of 2 RelayNode validation checking of realistic documents. In these cases, 1 core processing is a little more efficient.

Regarding system buffer access time (Figure 13 and 14), pipeline processing incurs larger buffer access time than parallel processing. In addition, most of 4 RelayNode processing incurs larger buffer access time than 2 RelayNode processing, because the amount of data passing through a node and assigned for process to the node is larger in the former case. In general, 1 core processing shows lower buffer contention than 4 core processing, as System buffer access time increases with more cores. Comparing the system buffer access time with other indicators, some indicators (e.g. system job execution time) are better for more CPU core processing but they also include inefficient performance because of useless waiting time caused by buffer contention.

Figures 15 and 16 further show node active time when processing doc01, for each node in the system.Generally, parallel processing is better than pipeline processing regarding node activity. This is because extra data transfer at each RelayNode is reduced in parallel processing, as compared with pipeline processing. Regarding the number of CPU cores in these figures, 4 core processing is better for StartNode, RelayNode and EndNode processing synthetic document. For MergeNode, 1 core performs better when processing synthetic documents. Across all node type, realistic document processing is superior when 4 cores are used.

Figures 17 and 18 further show node thread working time for processing kernel_doc. Generally, node thread working time is better for parallel processing using more RelayNodes. Regarding varying number of CPU cores, 4 core is better for StartNode when processing synthetic documents. However, 1 core RelayNode, EndNode and MergeNode present better node thread working time than their 4 core counterparts. Roughly, 1 core synthetic document processing is better in the following cases: less processing (e.g. well-formedness checking) and good document partition on RelayNodes; bad document partition, which leads to more processing for EndNode, for EndNode; Merge node, which leads to low buffer contention.

For convenience, we organize our performance characterization results about number of CPU cores

Table 3: Distributed XML Processing Characterization Summary.

	Synthetic docs (smaller docs)	Realistic docs (larger docs)
Job execution time	4 core is better	1 core is better in large proc.
System active time		4 core is better
System proc. time	1 core is better in WEL	Sometimes, 1 core is better in large proc.
Parallelism efficiency ratio	4 core is more efficient	4 core is more efficient in most cases
Sys. buffer access time	1 core is better	
Node active time	4 core is better in SN, RN and EN	4 core is better
Node thread working time	4 core is better in SN; Sometimes, 1 core is better in RN, EN and MN	4 core is better in SN; 1 core is better in EN; Sometimes, 1 core is better in RN and MN

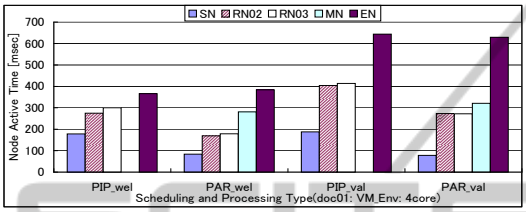

Figure 15: Node Ac. Time (doc01; 2RN; 4core).

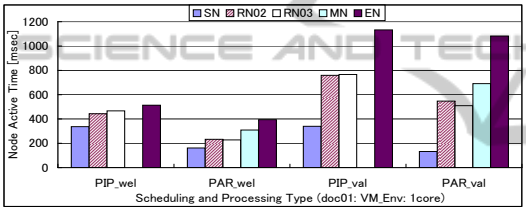

Figure 16: Node Ac. Time (doc01; 2RN; 1core).

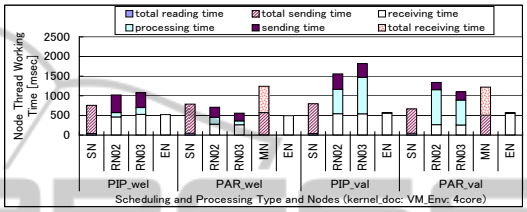

Figure 17: Node Th. Work. Time(kernel_doc; 2RN; 4core).

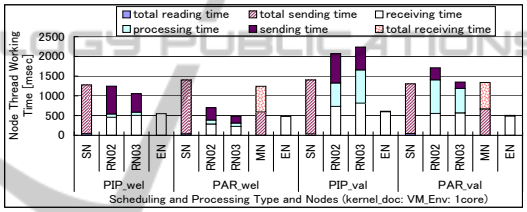

Figure 18: Node Th. Work. Time(kernel_doc; 2RN; 1core).

into Table 3 (WEL means well-formedness checking). A detailed analysis is also carried on at (Uratani et al., 2012), our previous work.

4 RELATED WORK

In this paper, we focus on offloading approach which assign a part of server's load to intermediate nodes in network for Web service. We here show some related works similar with our approach which have processing function at intermediate nodes. Active Network (Tennenhouse and Wetherall, 2007) is also focus processing at intermediate switches. The processing manner is described to two types: type-1) be capsuled into a packet, type-2) be assigned to the switches beforehand. The type-1 manner executes only simple processing but can process faster because of hardware execution. The type-2 manner, which like our system design, can describe even complicate processing. VNode(Kanada et al., 2012) also provides a processing environment at intermediate switches. In this research, the processing function is also provided at customized switches and its processing environment is provided as virtualized environment likewise using a virtual machine. These researches provide not only transport function but also processing function in the network. We address these intermediate processing model as a platform to achieve our distributed processing system.

Next we show current researches of processing in networks. In transcoding (Kim et al., 2012),a content server deliver data (e.g. video data) to clients via a transcoding server. The transcording server fixes the original data to another data which reflects user's demands. For instance, the data may be transformed from high resolution to low resolution at the transcoding server to adapt a mobile devices. A cache server(Nishimura et al., 2012; Kalarani and Uma, 2013) is a key technology of content delivery network. The cache servers are allocated to wide distributed places and they stock contents as cache from other content servers beforehand or with users request. Then the users' content request will lead to the nearest cache servers then the users receive contents with lower network latency. (Fan and Chen, 2012; Solis and Obraczka, 2006) focus sensor network. The researches propose to consolidate the large amount of sensing data at some intermediate nodes before the large data reach data collection servers.

Such approach can reduce energy consumption and network load for mobile sensor devices. (Shimamura et al., 2010) compress packets near a sender then extract the packets near a receiver during buffer queueing time to achieve low load network. In these researches and technology, we can assume the network provides special function such as video transforming function, data caching function and so on for a certain services.

5 CONCLUSIONS

In this paper, we have studied the impact of number of CPU cores in distributed XML processing using two models of distributed XML document processing in virtual environments for two types of XML document: parallel and pipeline models, on virtual machines with multicore CPUs, for synthetic and realistic XML documents. Regarding number of CPU cores for distributed XML processing, few CPU cores lead to less buffer contention. In contrast, more CPU cores leads to higher performance of some indicators, but with the drawback of incurring in wasteful buffer access waiting time. In addition, appropriate number of CPU cores depends on document characteristics. We can enhance the processing efficiency by improving buffer usage mechanism. As we have shown, pipeline processing is inefficient than parallel processing regardless document types and processing environment. The pipeline processing should treat parts of the document that are not to be processed at them. Such a specific node needs to be received and relayed to other nodes, consuming node resources and increasing processing overhead.

So far, we have focused on distributed well-formedness and validation of XML documents. These functions are a must for XML applications. The PASS-Node system guarantees the soundness of the XML document and it should lead to less battery consumption of mobile devices because of offloading. Moreover, other XML processing, such as filtering and XML transformations, can be studied. Internet routers in the future can do XML processing the same way routers today do deep packet inspection (Liu and Wu, 2013), as well as fast (hardware based packet) routing/forwarding.

We intend to study 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:24500043).

REFERENCES

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

Fan, Y.-C. and Chen, A. (2012). Energy Efficient Schemes for Accuracy-Guaranteed Sensor Data Aggregation Using Scalable Counting. *IEEE Transactions on Knowledge and Data Engineering*, 24(8):1463–1477.

Kalarani, S. and Uma, G. (2013). Improving the Efficiency of Retrieved Result through Transparent Proxy Cache Server. In *Proc. of fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT) 2013*, pages 1–8.

Kanada, Y., Shiraishi, K., and Nakao, A. (2012). Network-virtualization Nodes that Support Mutually Independent Development and Evolution of Node Components. In *Proc. of IEEE International Conference on Communication Systems (ICCS) 2012, pages 363–367.*

Kim, S. H., Kim, K., Lee, C., and Ro, W. (2012). Offloading of Media Transcoding for High-quality Multimedia Services. *Consumer Electronics, IEEE Transactions on*, 58(2):691–699.

Liu, C. and Wu, J. (2013). Fast Deep Packet Inspection with a Dual Finite Automata. *IEEE Transactions on Computers*, 62(2):310–321.

Nishimura, S., Shimamura, M., Koga, H., and Ikenaga, T. (2012). Transparent Caching Scheme on Advanced Relay Nodes for Streaming Services. In *Proc. of International Conference on Information Networking (ICOIN), 2012*, pages 404–409.

Shimamura, M., Ikenaga, T., and Tsuru, M. (2010). Advanced Relay Nodes for Adaptive Network Services - Concept and Prototype Experiment. In *Proc. of International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA) 2010*, pages 701–707, Los Alamitos, CA, USA.

Solis, I. and Obraczka, K. (2006). In-network Aggregation Trade-offs for Data Collection in Wireless Sensor Networks. *Int. J. Sen. Netw.*, 1(3/4):200–212.

Tennenhouse, D. L. and Wetherall, D. J. (2007). Towards an Active Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94.

Uratani, Y., Koide, H., Cavendish, D., and Oie, Y. (2012). Distributed XML Processing over Various Topologies: Characterizing XML Document Processing Efficiency. In *Web Information Systems and Technologies*, volume 101 of *Lecture Notes in Business Information Processing*, pages 57–71. Springer Berlin Heidelberg.