PGLC-XML: Um Paradigma baseado em GLC para a otimização da validação de arquivos XML

Thiago C. A. Sarmento[1], Eloi Favero[1], Otávio Onoura[1], Kelvin L. Dias[2]

[1] Instituto de Ciências Exatas e Naturais – Universidade federal do Pará (UFPA)
Caixa postal 479 – Belém – PA – Brasil
[2]Centro de Informática (CIn) – Universidade Federal de Pernambuco (UFPE)
Av. Jornalista Anibal Fernandes, s/n – 50.740-560 – Recife – PE – Brazil

thiagocarlossarmento@gmail.com, favero@ufpa.br, onoura@gmail.com, kld@cin.ufpe.br

Abstract: *XML files are currently used by all sorts of applications for promoting information exchange in an interoperable way, for example, the SOAP protocol used by Web Services. Thus, the parsing of this file is one of the biggest challenges with regard to resource consumption. An XML Parser is divided into four stages: parsing, access, modification and serialization. This paper proposes a new paradigm for implementing the first stage, by applying simplification rules of context free grammar. Comparison with traditional paradigm was carried out and the data were statistically analyzed: Student's t test applied to the average, and the correlation analysis was performed using the Pearson coefficient. The results showed significant differences between the paradigms evaluated. Our proposal, PGLC-XML (paradigm based on Context Free Grammar for optimization of the validation of XML files), is 85% faster than the traditional one, as well as it achieves a gain of 52% in terms of memory requirements. Furthermore, the throughput of PGLC-XML, in terms of number of operations, was 49% higher than the traditional paradigm.*

Resumo. *Arquivos XML são utilizados atualmente pelos mais variados tipos de aplicações para troca de informações de forma interoperável, como exemplo, pelo protocolo SOAP utilizado por Web Services. Assim, a análise (parsing) desse arquivo é um dos maiores desafios no que diz respeito ao consumo de recursos. Um Parser XML é dividido em quatro etapas: análise, acesso, modificação e serialização. Esse artigo propõe um novo paradigma para a implementação da primeira etapa, baseado na aplicação de regras de simplificação de gramática livre do contexto. Os dados foram analisados estatisticamente. Para o teste de média foi utilizado o teste t de Student e a análise de correlação realizada através do coeficiente de Pearson. Os resultados mostraram diferenças significativas entre os paradigmas testados sendo que o PGLC-XML (Paradigma baseado em GLC para a otimização da validação de arquivos XML) apresentou-se superior ao paradigma tradicional, pois este mostrou-se mais rápido 85%, obteve um ganho de memória de 52% e o maior número de operação em um dado tempo mostrando-se superior em 49%.*

1. Introdução

A *eXtensible Markup Language* (XML) vem sendo largamente utilizado por um número vasto de aplicações com objetivo de padronizar a representação das informações e a troca delas pela internet [Saigaonkar, Rao e Mantha, 2011]. Esse formato possui ainda vantagens no que diz respeito à interoperabilidade e é altamente utilizado como padrão para envio de informações através do protocolo SOAP implementado por *web services* [Salva e Rabhi 2010].

As qualidades do protocolo SOAP, bastante difundido e altamente interoperável, fazem com que esse protocolo seja largamente utilizado por aplicações de todos os fins, como por exemplo, transações bancárias que acessam banco de dados e aplicações que trabalham com monitoramento de vidas humanas [Lin et al. 2006]. Para esses tipos de aplicações, acredita-se que quanto menor o tempo de *parsing* dos arquivos utilizados nesse processo, mais eficiente serão as transações ou o tratamento de um paciente que é monitorado a distância, como por exemplo, aplicações que monitoram idosos que sofrem de demência sem interferir nas suas rotinas. Dessa forma *webservices* e o protocolo SOAP vêm sendo alvo de inúmeras pesquisas relacionadas à questão de desempenho, composição de mensagem e transferência comparado à outros protocolos [Zhang e Engelen 2008].

Com a larga utilização desse tipo de arquivo, diversos modelos de *parser* XML baseados nos modelos mais conhecidos na literatura SAX, STAX, DOM, VTD [Ding e Liu 2008] [Chang e Yu 2009] foram projetados e desenvolvidos com objetivo de otimizar a segunda etapa do processo, o acesso às representações de dados, em relação ao desempenho observando como métricas principais memória, processamento e *throughput*. Como exemplo, em [Mohd-Yasin e Mustapha 2009], onde é proposto um *parser* que implementa uma técnica de *rollback* chamado RBStrex baseado na junção dos modelos SAX e STAX e em [Chang e Yu 2009] que propõe um novo modelo de *parser* baseado no VTD.

Como o desenvolvimento da primeira etapa chamada *parsing* é invariante para todos os modelos, neste artigo é proposta uma forma de realizá-la aplicando regras de simplificação de gramática livre do contexto reduzindo duas etapas, análise léxica e sintática, utilizados no paradigma tradicional em apenas uma etapa. Dessa forma reduzindo o consumo de recursos como memória e processamento do processo do *parser* completo.

Isso ocorre, pois com o processo de simplificação objetiva-se aproximar a GLC ao máximo de uma Gramática Regular, no formato de uma Gramática Linear Unitária à Direita (GLUD). Nesse caso, a árvore de derivação de um programa passa a ter o formato de uma Árvore Binária e, assim, é possível armazenar em memória uma derivação, reconhecer o símbolo terminal e, depois, apagar o que foi processado. Teoricamente, passa-se a ter um uso constante do recurso de memória.

O presente artigo encontra-se dividido da seguinte forma: na Seção 2 são apresentados trabalhos relacionados. Uma breve descrição da primeira etapa desse processo, o *parsing* e a gramática livre do contexto são abordados na Seção 3. A Seção 4 detalha a descrição da proposta. O cenário e a implementação dos testes são descritos na Seção 5. Na Seção 6 são apresentados e discutidos os testes e resultados de desempenho obtidos. A conclusão e trabalhos futuros finalizam o artigo na Seção 7.

2. Trabalhos Relacionados

Com a larga utilização do formato XML nos mais variados tipos de aplicações, inúmeras pesquisas relacionadas ao *parser* são realizadas, porém como o *parser* é divido em quatro etapas e a primeira etapa chamada *parsing* é invariante para todos os modelos de *parser*, as pesquisas são voltadas em como projetar as estruturas de dados que receberão os valores validados pela primeira fase do processo e em como acessá-los[Ding e Liu 2008].

O trabalho apresentado por [Zhang e Engelen 2008] propõe o *table-driven streaming* XML *parsing*, denominado TDX. TDX análise do XML mais rapidanente através do pré-registo dos estados de um parser XML em forma tabular e utiliza em tempo de execução um mecanismo eficiente de análise de fluxo baseado em um autômato de pilha *(push-down* automaton). A análise de acordo com as tabelas são automaticamente produzidas a partir de esquemas XML de descrição de serviço WSDL(*Web Service Definition Language*).

Outra abordagem para *parser* XML é o chamado RBSTREX, que consiste em um algoritmo no nível de *hardware* que tem como propósito oferecer a dispositivos embarcados os mesmos serviços que são oferecidos a dispositivos com um alto poder computacional, porém com um menor consumo de memória e processamento. Esse algoritmo propõe uma nova funcionalidade ao *streaming parser* chamado SAX, funcionalidade essa chamada de *rollback*. Essa funcionalidade permite com que se faça o reuso de informações assim como é feito no DOM, VTD e STAX [Mohd-Yasin e Mustapha 2009].

Algumas abordagens utilizam *non-extractive parsing*, ou seja, modelo de *parsing* que mantém o conteúdo do arquivo texto intacto utilizando apenas os *offsets* e *lengths* para descrição dos *tokens*, diferente da abordagem *extractive parsing* que quebra os *tokens* do arquivo em objetos *string* [Zhang 2004]. Em [Chang e Yu 2009], propõe-se, utilizando *non-extractive parsing*, uma nova abordagem para análise desses arquivos, codificando informações de nó em números inteiros de 64 *bits* em vez de criar objetos de nó durante a análise, chamada NEM-XML.

Em [Lu, Chiu e Pan 2006], é proposta uma abordagem para realização de *parsing* XML de forma paralela que aproveita a prevalência crescente de arquiteturas de múltiplos núcleos em todos os setores do mercado de computadores. Essa proposta consiste de uma fase *preparsing* inicial para determinar a estrutura do documento XML, seguido por uma análise paralela completa. Como um trabalho seguinte, essa etapa de *preparsing* foi paralelizada com um mecanismo conhecido como meta-afd [Lu e Pan 2007].

Por outro lado, diferentemente das abordagens citadas anteriormente que utilizam o mesmo paradigma de validação de arquivos, essa proposta consiste em aplicar na primeira fase do processo de *parser*, um mecanismo de reconhecimento de estrutura baseada em uma Gramática Livre do Contexto Simplificada, diminuindo o consumo de memória e processamento desse processo como um todo.

3. *Parser* XML

Um *Parser* XML tem como objetivos principais, validar um arquivo de acordo com uma gramática pré-definida e colocá-lo em uma representação de dados para que assim os dados contidos no arquivo possam ser acessados por uma aplicação. Um *parser* é divido em quatro etapas, são elas: Análise, Acesso, Modificação e Serialização. A primeira etapa, alvo desse trabalho é dividida em três sub-etapas, conversão de caracteres, análise léxica e sintática, sendo estas definidas como padrão para validação de qualquer arquivo de texto.

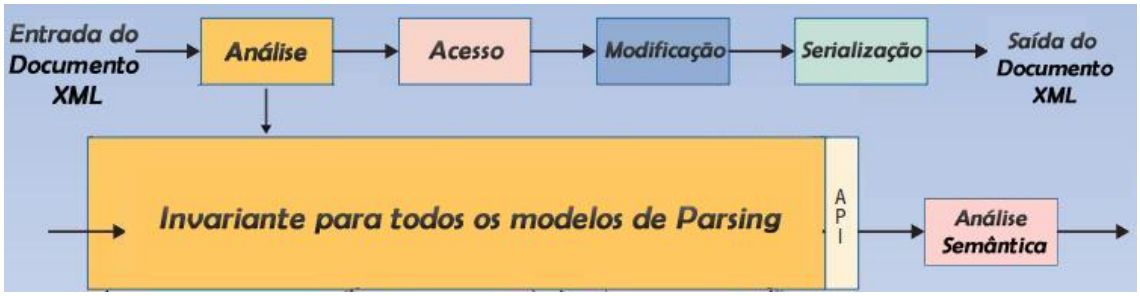

Figura 1. Processo do *parser* XML completo.

Na análise léxica espaços em branco e comentários são eliminados, em seguida uma máquina de estado finito (FSM) processa os caracteres um a um até encontrar os *tokens* que possuem um padrão definido por expressões regulares encontradas na especificação XML na W3C (*World Wide Web Consortium*). Em seguida, esses *tokens* são repassados para análise sintática que verificará se uma sequência definida é obedecida por uma gramática. Se o arquivo passar por essa segunda etapa, é então verificado por um autômato de pilha que contém a seguinte regra de transição descrita no pseudocódigo descrito no Algoritmo1, com a finalidade de validar o aninhamento correto das *tags*.

Algoritmo 1 : Algoritmo de pilha para aninhamento das Tags.

```
Inicio;
cadeiaDeToken[];
pAutomato.push("$");

while(cadeiaDeToken != vazio){
    token = cadeiaDeToken[];
    if(token == startElement){
        pAutomato.push(token);
    }
    else if(token == endElement){
        topo = pAutomato.top();

        if(top == endElement){
            pAutomato.pop();
        }
        else if(top == "$"){
        arquivo bem-formado;
        }
         else
          arquivo mal formado;
    }
}
```

Posteriormente a esta etapa, os *tokens* são colocados em representações de dados e estarão disponíveis para acesso e modificação por parte das aplicações por meio de APIs(*Application programming interface*) de desenvolvimento baseadas nos modelos de *parser* DOM, SAX, STAX e VDT. Como no paradigma padrão a etapa de validação é realizada em várias etapas, estruturas de dados são utilizadas para manter dados que serão reutilizados nas etapas seguintes.

3.1. Gramática Livre do Contexto

A Gramática Livre do Contexto (GLC) é um tipo de gramática na hierarquia de Chomsky chamada de gramática de Tipo 2 que tem a seguinte regra de produção:

Para G = (V, T, P, S), G: gramática, V: conjunto finito de variáveis, T: alfabeto composto por um conjunto de símbolos terminais, P: conjunto finito de regras de produção e S: símbolo inicial, onde:

$$A \rightarrow \alpha, \text{ tal que } A \in V \text{ e } \alpha = w(V \cup T)$$

Nessa gramática A deriva α que é independente de qualquer análise de símbolos que antecedem ou sucedem A, ou seja, é livre do contexto. Essa GLC quando simplificada ao máximo junta as fases de análise léxica e sintática em uma só leitura, tornando-se uma gramática Livre do Contexto sem Auto-Incorporação e consequentemente tornando-a em uma gramática regular linear a direita, com a seguinte regra de produção:

Para G = (V, T, P, S), onde

$$A \rightarrow wB$$

$$A \rightarrow w, \text{ tal que } A \text{ e } B \in V \text{ e } w \in T^*$$

Nessa Gramática, A deriva um Terminal seguido de um Não-Terminal e/ou somente um Terminal [Blauth 2008].

4. PGLC-XML

Para substituir as fases de Análise Léxica e Análise Sintática da implementação convencional, foi criada uma gramática G2 que contém regras de produção que, quando simplificada irá gerar uma GLC sem auto-incorporação.

$T \rightarrow TI\ T0$		$TI \rightarrow < i >$
$T0 \rightarrow ATF$	➔	$TF \rightarrow </ i >$
$A \rightarrow TI\ i\ TF \mid TI\ i\ TFA \mid \varepsilon$		$i \rightarrow (L+D)^*$

Figura 2. Gramática proposta para validação com o paradigma PGLC-XML, onde os *tokens tag* inicial é representado por TI, *tag* final por TF e qualquer caractere pela expressão regular em i, onde L+D: letra ou dígito. T, T0 e A representam a gramática em si com as possíveis combinações.

4.1. Simplificação da GLC Proposta

Com a GLC proposta tende-se a aplicar as técnicas de simplificação transformando a Gramática Livre de Contexto em questão em uma GLC simplificada sem Auto-Incorporação, com o intuito simplificar as etapas utilizadas no paradigma tradicional em apenas uma, assim otimizando o processo de *parser* XML como um todo, transformando a Gramática proposta em uma Gramática Regular.

- *Eliminação de Símbolos Vazios ε:* A primeira fase de simplificação é a eliminação de símbolos ε. Nessa fase, dada a gramática proposta G2 = (V, T, P, S), onde P segue em:

$$T \rightarrow TI\ T0$$
$$T0 \rightarrow ATF$$
$$A \rightarrow TI\ i\ TF \mid TI\ i\ TFA \mid \varepsilon$$

Figura 3. Gramática proposta para validação com PGLC-XML

- *Eliminação de Símbolos que Substituem Variáveis:* Aplicando a regra de eliminação de símbolos que substituem variáveis a partir da gramática G2, se tem a nova gramática G2'' com as seguintes produções:

$$T \rightarrow < i >T0$$
$$T0 \rightarrow < i > i </ i > </ i > \mid < i > i </ i > A </ i > \mid </ i >$$

Figura 4. Eliminação de símbolos inúteis.

- *Fatoração:* Aplicando a regra de fatoração a partir da gramática G2'', se tem a gramática G2''' regular final com as seguintes produções:

$$T \rightarrow < T_1 \qquad T_4 \rightarrow > T_7 \qquad T_9 \rightarrow / T_{10}$$
$$T_0 \rightarrow < T_3 \qquad T_5 \rightarrow i\ T_6 \qquad T_{10} \rightarrow i\ T_{11}$$
$$T_1 \rightarrow i\ T_2 \quad \rightarrow \qquad T_6 \rightarrow > \quad \rightarrow \qquad T_{11} \rightarrow > T_0$$
$$T_2 \rightarrow > T_0 \qquad T_7 \rightarrow i\ T_8$$
$$T_3 \rightarrow i\ T_4 \mid /T_5 \qquad T_8 \rightarrow < T_9$$

Figura 5. Fatoração da gramática tornando-a uma gramática livre do contexto simplificada.

- *Árvore de Derivação da GLC Simplificada:* Devido a estrutura da gramática simplificada ser regular [Blauth 2008], essa gramática exige um poder de processamento menor e também um menor uso de memória, pois ela gera uma árvore binária na derivação, ou seja, o processamento dos caracteres e estados gerados pela gramática são sempre processados agrupados de 2 em 2, sempre "consumindo" um caractere seguido de um estado na gramática. Esse tipo de processamento exige que a memória seja utilizada em "fatias" previamente

alocadas e fixas, diminuindo o custo tanto de utilização de memória quanto de processamento.

5. Metodologia de Avaliação da Proposta

Para realização dos testes foram analisadas três métricas, são elas: o tempo de processamento, memória e o *throughput* que foi definido em número de operações a cada vinte segundo. As métricas descritas foram analisadas por um *benchmark* desenvolvido em Java, baseado na implementação de *benchmark* XML descrito em (Ximpleware 2012). Para analisar o consumo de memória classes Java como *Runtime* que estende *Object* e métodos como *totalMemory()* e *freeMemory()* foram utilizados, para o tempo de processamento a classe *System* e seu método *currentTimeMillis()* e para o *througput* foi implementado um algoritmo que mostra ao final de um tempo determinado no código quantas validações ou operações foram realizadas nesse intervalo de tempo. Para todas as métricas foram instanciadas duas *Threads* cada uma responsável pela validação de uma parte do arquivo com o objetivo de se obter uma validação paralela. A seguir será mostrado com mais detalhes como foram realizados os testes e a implementação do *benchmark* para cada métrica.

O *benchmark* para a medição de dados sobre o consumo de memória utilizou dois métodos: *totalMemory(),* que retorna em *bytes* a quantidade total de memória na JVM *(Java Virtual Machine)* e *freeMemory(),* que retorna a quantidade de memória livre até a execução do mesmo. Antes de executar a linha que inicia o algoritmo que representa o paradigma proposto é executada a seguinte expressão:

$$memoriaInicial = (memoriaTotal - memoriaLivre)$$

memoriaInicial será a quantidade de memória utilizada antes que o algoritmo principal seja iniciado. Depois que esse algoritmo é finalizado e o arquivo é validado a seguinte expressão é executada:

$$memoriaFinal = (memoriaTotal - memoriaLivre)$$

memoriaFinal é a quantidade de memória utilizada no final da validação do arquivo. Após essas duas variáveis serem obtidas, outra expressão é calculada para se obter a quantidade de memória utilizada somente durante a validação do arquivo, ou seja, durante o processamento do algoritmo principal. A expressão executada é:

$$memoriaTotalConsumida = (memoriaFinal - memoriaInicial)$$

memoriaTotalConsumida é a quantidade de memória que se consumiu somente durante o processamento do algoritmo principal, esse valor é convertido em seguida de *bytes* para *megabytes* pois essa grandeza é a utilizada na análise estatística que será mostrada na próxima seção.

O *benchmark* utilizado na medição de dados para tempo de processamento utilizou a classe *System* e seu método *currentTimeMillis()* que retorna o tempo decorrido da execução do *benchmark* até o processamento desse método, em milissegundos. Antes de executar o algoritmo principal para validação do arquivo foi realizado a coleta do *tempoInicial* decorrido da execução do benchmark até a linha que antecede o inicio da validação. Em seguida a mesma é realizada e após isso é coletado o *tempoFinal* que corresponde ao tempo decorrido da execução do *benchmark* até a validação completa do

arquivo. O último passo para se obter somente o valor do tempo da validação é calcular a seguinte expressão:

$$tempoTotalValidacao = (tempoFinal - tempoInicial)$$

O *benckmark* utilizado na medição do *througput* em operações por segundo utilizou um algoritmo implementado a classe *System* e seu método *currentTimeMillis()* e é mostrado a seguir:

Algoritmo 2: Algoritmo para coleta de dados relacionados ao Throughput.

```
long tempoInicial = System.currentTimeMillis();
        long constante = 20000;
        int cont = 0;

        while(true){
            chamada do método de validação
            if(((System.currentTimeMillis() - tempoInicial) > constante)){
                break;
            }
            cont ++;
        }
    System.err.println(cont);
```

Onde "constante" é o tempo total em milissegundos que o arquivo será colocado em análise, dessa forma, será realizada N validações em vinte segundos decorridos.

Os testes foram realizados validando um arquivo XML com 2.044 linhas no paradigma tradicional que utiliza basicamente conversão de caractere, análise léxica e sintática como um padrão de validação de arquivos de texto e no paradigma proposto nesse artigo que implementa regras de simplificação de gramática livre do contexto simplificando assim os dois passos análise léxica e sintática em apenas um.

No paradigma tradicional o arquivo XML foi validado de forma sequencial por inteiro, sendo que no paradigma proposto o mesmo arquivo foi divido em duas partes iguais, para que então duas *Threads* independentes pudessem validar as duas partes separadamente e em paralelo. O resultado dos testes será mostrado e discutido a seguir.

Os testes foram executados em um computador com Windows7, 4GB de Memória e Processador Core2 Duo 2.8 GHz. A comparação das médias dos paradigmas foi realizada por meio do teste t de *Student* a 5% de probabilidade. Os intervalos de confiança (CI) obtidos entre os dois paradigmas foram calculados pela fórmula descrita na Figura 1. Foi utilizado o delineamento inteiramente casualizado (Pimentel Gomes, 2000) com 40 amostras de cada paradigma para as três métricas. E, ainda, foi feita a análise de correlação pelo coeficiente de Pearson a 5% de probabilidade. Para a análise dos dados foi verificado o coeficiente de variação e o desvio padrão para cada paradigma. Foi utilizado o Software estatístico SPSS versão 16.0.

$$\left((\bar{X}_1 - \bar{X}_2) - z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}, (\bar{X}_1 - \bar{X}_2) + z_{\alpha/2}\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \right)$$

Figura 6. Expressão do Intervalo de confiança entre as médias dos dois paradigmas.

6. Resultados e discussão

Os resultados indicam que os paradigmas discutidos nesse trabalho tiveram pouca variabilidade e dispersão nos dados como é observado na Tabela 1, demonstrando a confiabilidade dos mesmos. Pode-se perceber que os desvios-padrão da memória e do *throughput* foram menores no paradigma tradicional do que no PGLC-XML, significando que possuem menor dispersão dos dados. Ainda nessa tabela, pode-se observar baixo coeficiente de variação das variáveis analisadas mostrando dessa forma a homogeneidade dos dados.

Tabela 1. Média, D.P. (desvio-padrão) e CV (coeficiente de variação) das variáveis memória, tempo e *throughput* analisadas em dois paradigmas, tradicional e PGLC-XML.

Variáveis	Memória		Tempo		*Throughput*	
Paradigmas	PGLC-XML	Tradicional	PGLC-XML	Tradicional	PGLC-XML	Tradicional
Média	0.7577	1.5627	0.0645	0.4091	1806.6154	878.4615
D.P.	0.0430	0.0026	0.0023	0.0139	27.1822	9.4642
CV (%)	5.6742	0.1681	3.5162	3.3888	1.5046	1.0774

O teste t (Tabela 2) rejeitou o H0 cuja a hipótese é nula, para as variáveis de tempo, memória e *throughput*, significando que as médias das mesmas são diferentes entre si para os paradigmas testados. Contudo, observa-se estatisticamente com 5% de probabilidade (Figura 6) que as médias de tempo e de memória no PGLC-XML são menores do que o tradicional, sendo 85% mais rápido, obtendo 52% de ganho em relação à memória, e que o *throughput* é superior executando 49% mais operações.

Tabela 2. Teste t de *Student* da memória, tempo e *throughput* para os dois paradigmas de validação testados.

	Memória	Tempo	*Throughput*
p- valor	0.0000	0.000	0.000
Conclusão	rejeita H0	rejeita H0	rejeita H0
Cv	2.61	4.14	1.5

As análises de correlação ($p<0,05$) os coeficientes de *Pearson* mostram (Tabela 3) que a memória e o tempo correlacionam-se positivamente indicando que com o aumento no consumo de memória há o aumento do tempo, e o *throughput* apresenta correlação negativa com a memória e o tempo, ou seja, com o aumento da memória ocorre a diminuição do *throughput*, o mesmo acontecendo para o tempo.

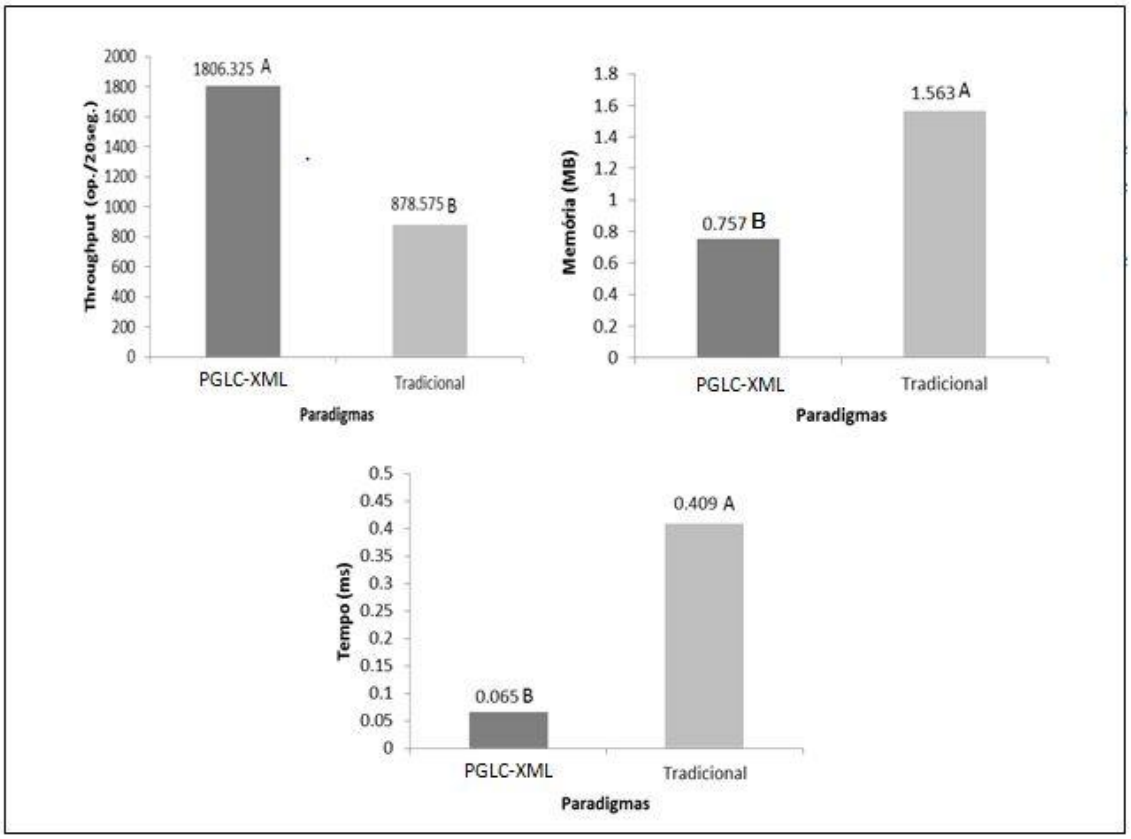

Figura 7. Tempo, memória e *throughput* avaliados nos paradigmas, tradicional e PGLC-XML, para validação dos arquivos XML. Mesmas letras significam médias iguais pelo teste t de *Student* com $p<0,05$.

Tabela 3. Estimativas dos coeficientes de correlação de *Pearson* entre as variáveis de memória, tempo e *throughput* avaliados nos paradigmas testados.

	Memória	Tempo	*Throughput*
Memória	-		
Tempo	0,996**	-	
Throughput	-0,996**	-0,997**	-

** Correlação significativa com 0,01% utilizando o coeficiente de correlação de *Pearson*.

7. Conclusão e trabalhos futuros

Parser XML é um processo realizado em vários segmentos da computação, no contexto de *web services* e atua como uma fase do seu processamento completo, sua importância consiste nas aplicações que dependem de um tempo de resposta rápido tendo essa métrica como um requisito de alta importância para a sua execução correta, muitas vezes aplicações que trabalham com situações críticas como a vida humana e

outras. Dessa forma, quanto menor for o tempo de processamento das mensagens mais satisfatório será o funcionamento dessas aplicações.

Os resultados mostram que a aplicação do PGLC-XML reduz o tempo de processamento e consumo de memória, demonstrando assim a eficácia desse paradigma.

Para trabalhos futuros, pretende-se aplicar esse paradigma a dispositivos móveis como celular e Sun SPOT para obter melhor desempenho, pois esses dispositivos têm de baixo poder computacional.

Referências

Blauth, P. (2008), Linguagens Formais e Autômatos, bookman, 5ª Edição.

Ding, J.J. e Liu, J-C. (2008) "XML Document Parsing: Operational and Performance Characteristics", Quality Software (QSIC), 10th International Conference.

Zhang, J. (2004) "Non-Extractive Parsing for XML", http://www.xml.com/pub/a/2004/05/19/parsing.html, Novembro.

Lin, C.C.; Chiu, M.J.; Hsiao, C.C.; Lee, R.G.; Tsai, S. (2006) "AWireless Healthcare Service System for Elderly with Dementia". Information Technology in Biomedicine, IEEE Transactions on, v.10, n.4. p. 696-704.

Lu, W., Chiu, K. e Pan, Y. (2006) "A Parallel Approach to XML Parsing", Grid Computing, 7th IEEE/ACM International Conference.

Lu, W. e Pan, Y. (2007) "Parallel XML Parsing Using Meta-DFAs", e-Science and Grid Computing, IEEE International Conference.

Yasin, F. e Mustapha, A.K. (2009) "RBStreX: Hardware XML parser for embedded system", Internet Technology and Secured Transactions, ICITST. International Conference.

Saigaonkar, S., Rao, M. e Mantha, S. (2011) "Publish Subscribe System Based On Ontology and XML Filtering", Computer Research and Development (ICCRD), 3rd International Conference.

Salva, S. e Rabhi, I. (2010) "Stateful Web Service Robustness", Internet and Web Applications and Services (ICIW), Fifth International Conference.

Zhang, W. e Engelen, R. (2008) "An Adaptive XML Parser for Developing High-Performance Web Services", eScience '08. IEEE Fourth International Conference.

Chang, Y. e Yu, L. (2009) "NEM-XML: A Fast Non-extractive XML Parsing Algorithm", Multimedia and Ubiquitous Engineering, Third International Conference.

PIMENTEL GOMES, F. Curso de estatística experimental. 14ª ed. Piracicaba – SP: Editora da Universidade de São Paulo, 2000. 477p.

XIMPLEWARE. 2012. XML Parsing Performance Benchmark of VTD-XML 1.5. Disponível em http://www.ximpleware.com/benchmark1.html. Acesso em 10 de abril de 2012.