Modelo de Recuperação Arquitetural QoE-QoS Híbrido para Bases de Dados Distribuídas Gerenciado por Middleware

Ramon Hugo de Souza, Mario Antonio Ribeiro Dantas

Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis - SC Brazil

ramon@securebay.com, mario.dantas@ufsc.br

Resumo. *O conceito de qualidade de experiência (QoE) não é uma novidade per se, mas sua definição em termos de métricas mapeáveis para qualidade de serviço (QoS), e em especial tal abordagem aplicada ao contexto de bases de dados distribuídas (BDDs), é um novo conceito. Novos trabalhos formalizando QoE como completude de QoS permitem se lidar com decisões diretamente relacionadas a correção de desempenho em nível de sistema, não mais considerando-se o conceito abstrato de QoE, usualmente ligado a satisfação de usuário com base em avaliações tácitas. Esta pesquisa tem como objetivo a demonstração do ganho temporal que pode ser obtido em considerações de replicação com gerenciamento via a abordagem de um middleware.*

1. Introdução

O conceito de *qualidade de serviço* (*QoS*) é bem fundamentado na área de *redes*, origem de suas definições em *ciência da computação*. Sendo o mesmo apresentado em [Wolter and van Moorsel 2001] como o conjunto de métricas de vazão, atraso e disponibilidade, é um conceito apresentado de maneira mais rígida em [Shenker et al. 1997] com a definição utilizada para *serviço garantido*, a qual firma que o mesmo "provê limites firmes e matematicamente prováveis de atraso fim-a-fim em enfileiramento de datagramas".

A definição conceitual de *QoS* em termos de *métricas matematicamente prováveis*, a qual pôde-se abstrair além da fundamentação de datagramas de redes, foi o passo fundamental que permitiu a aplicação de tal conceito a abstração de *bases de dados distribuídas* (*BDDs*) nos primeiros trabalhos sobre o assunto, publicados em meados dos anos 2010.

Além de tal, tem-se também a definição do conceito de *qualidade de serviço* aparecendo em diversos artigos publicados junto a área de *marketing* [Lewis and Booms 1983, Gronroos 1984, Parasuraman et al. 1985, Parasuraman et al. 1988, Parasuraman et al. 1994], onde o mesmo pode ser concisamente definido como "o resultado da comparação que clientes realizam entre suas expectativas sobre o serviço e sua percepção sobre a maneira como o serviço foi realizado" [Caruana 2002]. Sendo esta na verdade a conceitualização utilizada para *qualidade de experiência* (*QoE*), e não *QoS*, em *ciência da computação*, com suas origens provavelmente advindas destes conceitos de *marketing* qualificando a experiência do usuário sobre um serviço consumido.

Esta conexão com tais conceitos de *marketing* pode ser visualizada na definição dada em [ISO9241-210 2010], onde é definido que a *experiência do usuário* envolve "as

percepções de uma pessoa e suas respostas resultantes do uso, ou antecipação do uso, de um produto, sistema ou serviço".

Na explicação de fundo teórico da conexão entre *QoE* e *conceitos de QoS* dada em [ur Rehman Laghari et al. 2011], tem-se que *QoE* pode ser considerada como um mapeamento direto, porém ainda *difuso*, sobre como um usuário percebe um serviço em termos de *parâmetros de QoS*. Sendo citado como digno de nota em [de Souza and Dantas 2015] que "uma vez que os *parâmetros de QoE* são baseados em comportamento e expectativas humanas, é difícil se assumir o nível de precisão neste processo de mapeamento. Um problema que desaparece quando se considerando sistemas ao invés de usuários".

E então tem-se uma primeira definição que permite se lidar com avaliações de *QoE* em sistemas de *BDDs* quando se definindo métricas palpáveis de *QoS* em termos de médias e desvios padrão - considerado-se a combinação destes parâmetros como satisfatória para definição de métricas matematicamente prováveis sobre garantias dadas.

Na seção 2 apresenta-se: (i) a origem dos conceitos de *QoE* aplicados a *BDDs*; (ii) um primeiro modelo lidando com avaliação de base estritamente temporal; e (iii) um segundo modelo lidando com avaliação diretamente sobre completude de recursos. Na seção 3 é apresentado o novo modelo com gerenciamento de recursos utilizando um *middleware*, com os resultados de simulação demonstrados na seção 4. Finaliza-se com as conclusões e trabalhos futuros junto a seção 5.

2. QoE em BDDs e Modelo Arquitetural de Recuperação

2.1. QoE em Bases de Dados Distribuídas

No trabalho de levantamento bibliométrico [de Souza and Dantas 2015] é discutida a ausência de definições precisas de *qualidade* para *BDDs* num período de levantamento realizado no ano de 2014 e ainda válido em 2017. Neste trabalho é pontuado que conceitos em literatura focada em redes possuem natureza por demais holística, o que levou tal a lidar com definições conceituais para demonstração da importância de *QoE* para *BDDs*.

Sendo também demonstrado que uma melhor definição de *QoS* em termos de médias e desvios padrão de crucial importância para quando se lidando com garantias de *QoE*, seja para *BDDs* ou mesmo tecnologias relacionadas a redes. Formalizando assim métricas matematicamente prováveis utilizadas em trabalhos subsequentes.

2.2. Modelo Arquitetural de Recuperação Focado em RAS

Em [de Souza et al. 2016] é apresentado um primeiro modelo de recuperação baseado na chamada *capacidade RAS*[1] *(confiabilidade/disponibilidade/facilidade de manutenção do inglês reliability/availability/serviceability)*: com restrições de confiabilidade, disponibilidade e facilidade de manutenção - incluindo restrições de tolerância a falhas.

Sendo este modelo completamente baseado em verificação de restrições temporais, gerando o empecilho de se mapear medidas de recursos para seu efeito temporal.

[1]RAS: Um termo originalmente utilizado pela IBM para descrever a robustez de seus computadores *mainframe* [Siewiorek and Swarz 1998].

2.3. Modelo Arquitetural Estendido

Em [de Souza and Dantas 2017] é apresentada uma primeira solução para se evitar o mapeamento de recursos para sua representação temporal, sendo um primeiro modelo baseado somente em avaliação de recursos provistos.

Em tal artigo é apresentado um modelo lidando com dois anéis completos e tolerância a uma falta na comunicação entre anéis e na comunicação entre os nós replicantes no que é chamado de *pool provedor de serviços*. É apresentado também um *pool de backup* com um semi-anel de réplicas dormentes.

No algoritmo 1 é apresentado o procedimento principal de tal modelo, que é utilizado em explicação comparativa na próxima seção sobre o porquê da mudança em processo decisório.

Algoritmo 1 Procedimento principal do algoritmo de gerenciamento de recursos em sistemas distribuídos

1: **procedimento** $Principal(tempo, tipo, passo)$
2: **enquanto** $verdade$ **faça**
3: **se** $\frac{\sum_{i=1}^{\#replicas} replicas[i].c_{livre}}{\#replicas} \leq ANS^{min}_{RASGarantia}$ **então**
4: call $RAS_{min}(tempo, tipo, passo)$
5: **senão se** $c - 2 * ANS^{min}_{RASGarantia} \geq \frac{\sum_{i=1}^{\#replicas} replicas[i].c_{usado}}{\#replicas}$ **então**
6: call $RAS_{max}(tempo, tipo, passo)$
7: **fim se**
8: $espere(tempo)$
9: **fim enquanto**
10: **fim procedimento**

Este é um procedimento que checa constantemente se o valor mínimo de recursos definido em *Acordo de Nível de Serviço* (*ANS*), $ANS^{min}_{RASGuarantee}$, é garantido, chamando o procedimento de correção $RAS_{min}(tempo, tipo, passo)$ quando tal valor é alcançado. Sendo este procedimento de correção responsável por iniciar mecanismos que visam prover mais recursos.

O procedimento principal checa também se é possível se reduzir a capacidade de recursos provistos, chamando o procedimento responsável $RAS_{max}(tempo, tipo, passo)$ caso a redução de capacidade não apresente riscos ao sistema em relação a garantia dada sobre o valor definido em ANS.

Tal algoritmo trabalha com quatro modos de operação: (i) otimista: baseado no algoritmo apresentado em [de Souza et al. 2016], mas com mensuramento de recursos e não avaliações temporais; (ii) não-otimista: uma versão em contrapartida ao modo otimista; (iii) balanceado: uma tentativa de balanceamento entre os dois primeiros modos; e (iv) não-otimista otimizado: a evolução do modo não-otimista focada em economia de recursos.

Note, na linha 8, que este algoritmo tem como consideração um valor de tempo de espera como intervalo de verificação de se os recursos podem ser modificados.

3. Modelo Arquitetural Estendido Gerenciado por Middleware

O maior empecilho encontrado no algoritmo do modelo arquitetural estendido, apresentado na seção anterior, é a necessidade de verificação temporal. Tal verificação não é pontual, e o sistema lidando com passos discretos pode passar por estados indesejados desnecessariamente. Esta verificação temporal é legado do algoritmo apresentado em [de Souza et al. 2016], o qual tem por base avaliação somente temporal.

Na figura apresentada em tal artigo percebe-se uma pequena falha de consideração na representação gráfica da arquitetura quando se verificando o procedimento no algoritmo 1. De maneira que quem realmente deve realizar as chamadas de incremento ou decremento de recursos é o *middleware*, tanto no caso vertical quanto no horizontal.

E então o *stress* de replicação aplicado sobre os nós $n04$ ou $n07$ - a figura 1 apresenta um modelo com as mesmas especificações - deixa de existir. Note também que este *stress* de replicação não foi demonstrado na simulação apresentada em [de Souza and Dantas 2017].

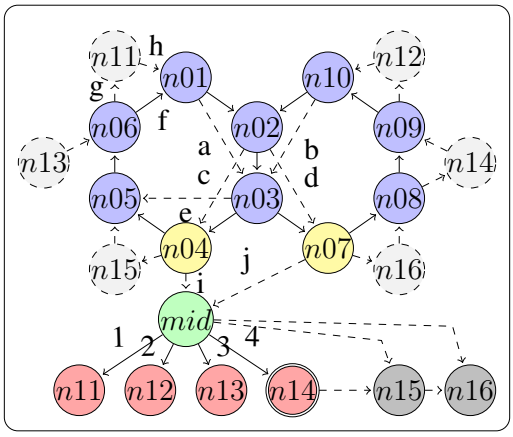

Figura 1. Modelo arquitetural estendido gerenciado por middleware com implementação de troca de mensagens utilizando o modelo multi-anel e tolerância a uma falta.

Para demonstração deste efeito de *stress* de replicação considera-se o consumo de uma unidade de tempo sobre o nó $n04$, responsável pela replicação, e sobre o nó a ser adicionado utilizando os quatro modos - nas figuras 3, 5, 7 e 9 com as mesmas considerações de simulação publicadas no referido artigo sem o *stress*. O efeito de *stress* de replicação é, em geral, muito maior que este simulado, mas para demonstração considerar-se-á que ocorre a sincronização de operações somente no momento de adição de nó. Note que este é o melhor cenário possível, ocupando apenas um tempo dos nós e somente quando um nó é adicionado ao *pool provedor de serviços*.

De maneira que a representação do modelo, evitando-se o *stress* de replicação sobre os *nós de borda*[2], deve ser gerenciada via *middleware*, como apresentado no modelo da figura 1.

[2]Nós referenciados em [de Souza and Dantas 2017] como os responsáveis pela gerência de replicação junto ao *pool de backup*.

Onde então os nós $n04$ e $n07$ não mais precisam armazenar uma sequência especial de operações a ser enviada as réplicas no *pool de backup*, somente enviando uma cópia das operações em seu *buffer* ao *middleware*, da mesma maneira como o nó $n04$ envia ao nó $n05$ em sua sequência de atualização padrão.

Note que neste modelo gerenciado por *middleware* a replicação deve ser *sequencial completa*, e não transação a transação, afinal deseja-se obter uma réplica funcional o mais rápido possível caso seja necessária adição ao *pool provedor de serviços*, mesmo durante a fase de replicação - a replicação poderia estar acontecendo junto ao nó $n12$ enquanto o nó $n11$ já estaria pronto para ser adicionado caso necessário.

Neste novo modelo tem-se também uma consideração especial quanto ao *último nó* parte do *pool de backup*. No exemplo apresentado na figura 1 tem-se que no momento em que o nó $n11$ é adicionado ao *pool provedor de serviços* o *middleware* acorda o nó $n14$ para se responsabilizar pela obtenção de dados necessária a instanciação do novo nó $n15$, a ser adicionado ao *pool de backup*. De maneira a explicitarmos que não é o nó $n14$ que se responsabiliza pela cópia, e sim o *middleware*.

4. Resultados Experimentais

Nesta seção apresenta-se os resultados relativos aos experimentos, visando demostrar o diferencial da proposta estendida.

Quanto aos modos de operação, tem-se inicialmente o modo otimista, que já fora demonstrado "problemático" em sua primeira utilização junto ao modelo de avaliação temporal. Na figura 2 pode-se verificar tal impacto sobre os valores de recursos provistos ainda desconsiderando-se o *stress* de replicação.

Note que este modo já apresenta o comportamento peculiar de lidar com picos abaixo da capacidade esperada, podendo-se observar o comportamento do mesmo com a consideração de *stress* de replicação junto a figura 3.

Quando lidando-se com o modo otimista fica claro que o efeito do *stress* de replicação é uma consideração que pode trazer sérias implicações quanto a utilização deste tipo de arquitetura, em um modo o qual já apresentava menos recursos que o necessário quando da replicação horizontal sem a consideração de tal efeito.

Os modos não-otimista, balanceado e não-otimista aprimorado apresentam comportamento estruturalmente parecido quando da replicação horizontal, mas mesmo não sofrendo como o modo otimista todos eles apresentam momentos em que a capacidade requerida não é provista pelo sistema a tempo. De maneira que fica claro que as considerações feitas no modelo anterior, sem considerar efeitos de manutenção quando não realizados estritamente pelo *middleware*, podem ser realmente problemáticas.

No comparativo entre as figuras 4, 6 e 8 com seus equivalentes com a inserção do efeito de *stress* de replicação, nas figuras 5, 7 e 9, respectivamente, nota-se que a consideração pontuada no artigo original pode ser *não realista*, de maneira que a existência do *stress* de replicação causa situações em que os recursos não são provistos como previsto em tal modelagem. Um problema que desaparece ao se considerar o *middleware* como o responsável por tal processo de replicação.

Os modos não-otimistas possuem uma queda menor mesmo em consideração de

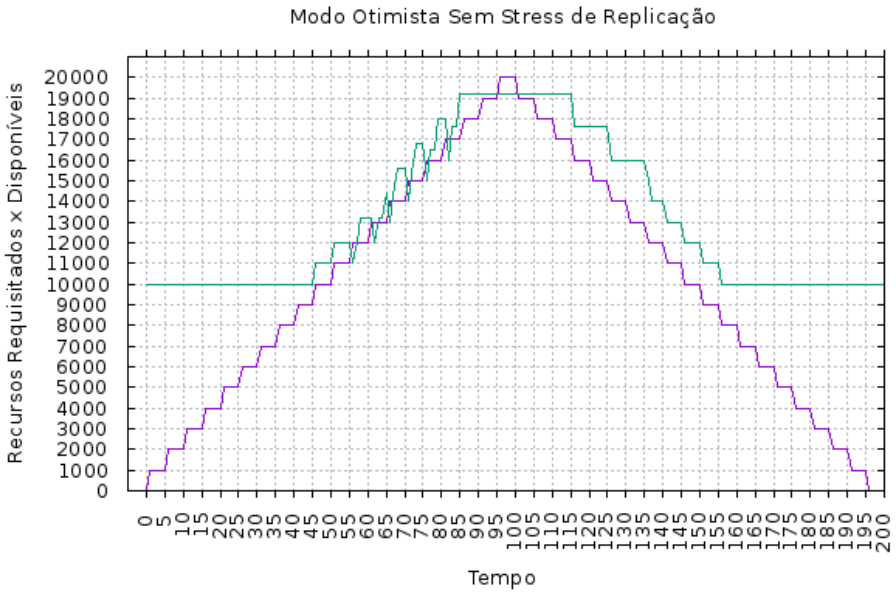

Figura 2. Modo otimista sem stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

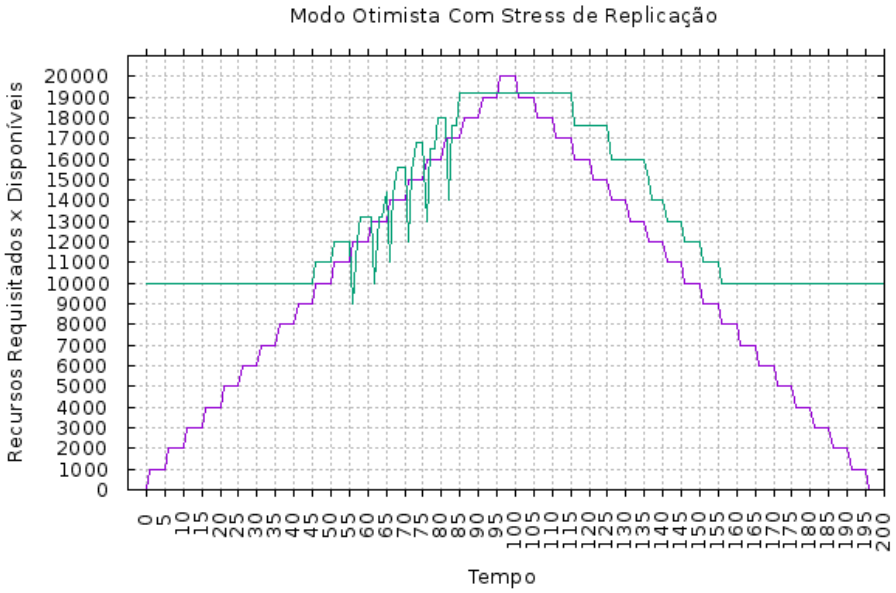

Figura 3. Modo otimista com stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

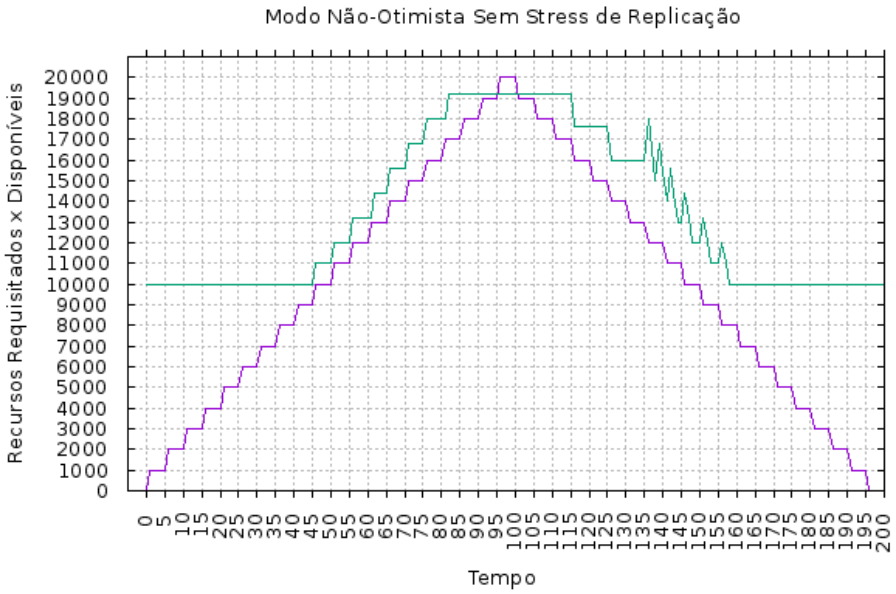

Figura 4. Modo não-otimista sem stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

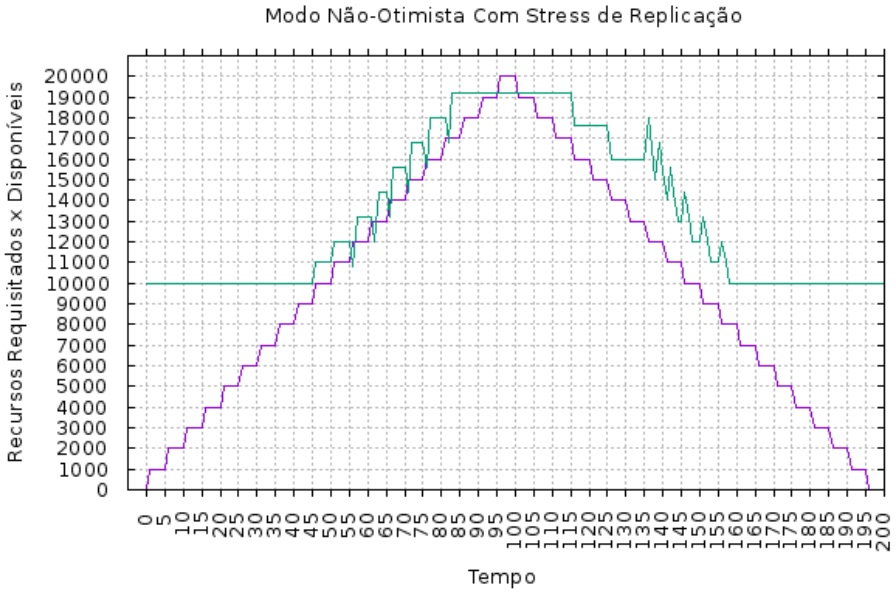

Figura 5. Modo não-otimista com stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

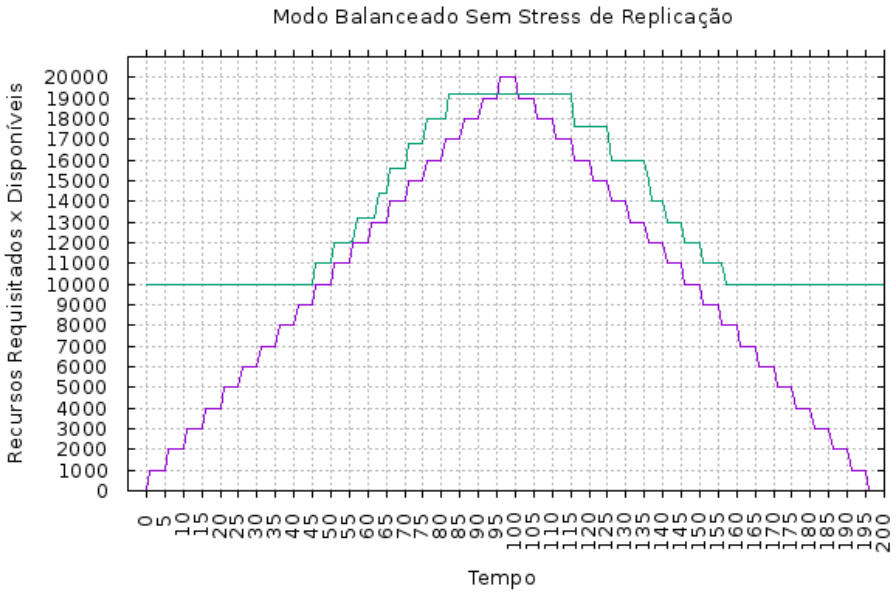

Figura 6. Modo balanceado sem stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

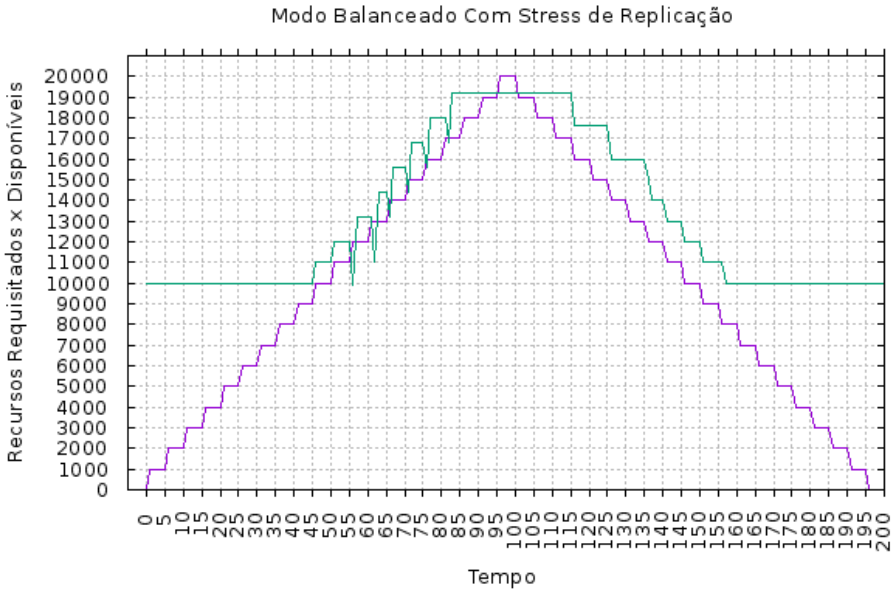

Figura 7. Modo balanceado com stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

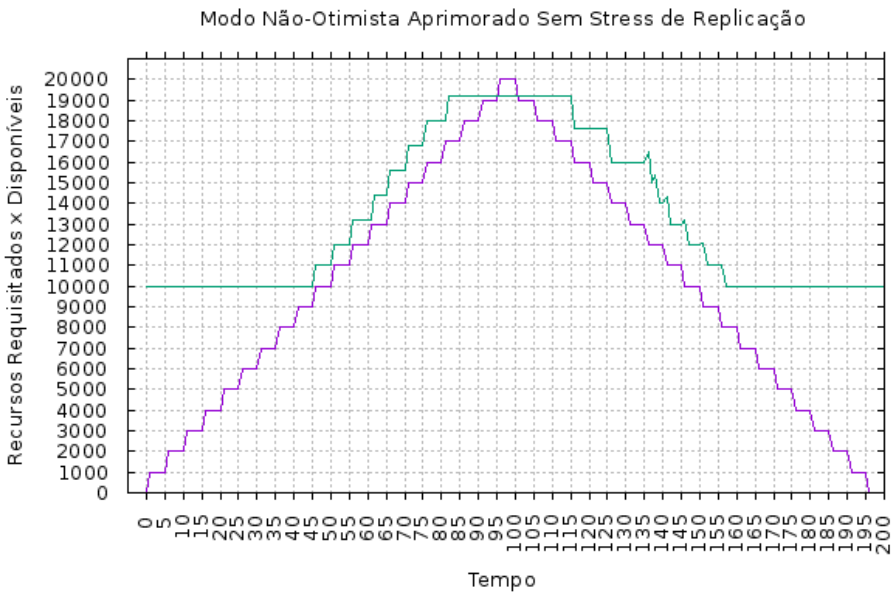

Figura 8. Modo não-otimista aprimorado sem stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

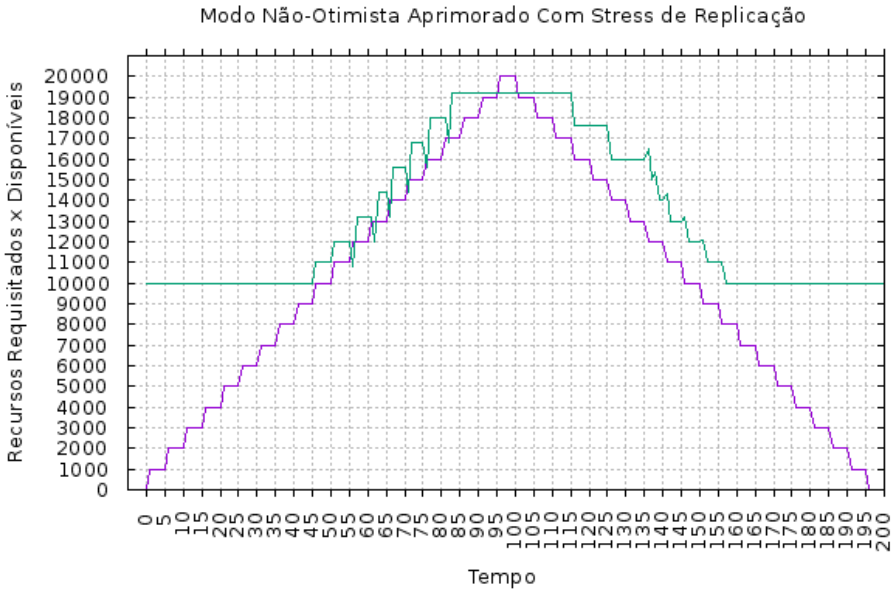

Figura 9. Modo não-otimista aprimorado com stress de replicação: Recursos requisitados (em roxo, iniciando em 0) contra disponibilizados (em ciano, iniciando em 10000).

stress de replicação, sendo o efeito sobre os dois modos de mesma magnitude. Tal efeito sobre o modo balanceado se encontra entre os modos não-otimistas e o modo otimista, sendo o modo otimista o mais afetado pelo efeito de *stress*, afinal o mesmo já possuía o problema de redução de recursos quando da replicação horizontal que dá nome ao modo.

E então apresenta-se o algoritmo 2 com as definições procedurais necessárias para que o *middleware* lide com este processo de replicação.

Tal algoritmo sendo somente um procedimento de adição de operações, a ser recebido pelo *middleware*, com o procedimento $Principal$, apresentado no algoritmo 1, sendo executado ao final do procedimento de adição de operações. Não existindo mais o a chamada a $espere(tempo)$ e sendo o intervalo de atualização agora o mesmo de atualização de mensagens entre nós.

Neste novo modelo os parâmetros são considerados globais, de maneira que os procedimentos RAS_{min} e RAS_{max} não necessitam mais de parâmetros em sua chamada. Para tais procedimentos agora considera-se também explicitamente a exclusão de seu loop de verificação com o intervalo temporal realizado pela chamada a $espere(tempo)$, excluindo-se completamente as considerações temporais e ainda assim mantendo válidos os valores simulados, afinal o tempo de espera simulado é justamente o intervalo de atualização de mensagens.

No algoritmo são explicitadas as operações *aloca(réplica)*, linha 19, e *desaloca(réplica)*, linha 26, para explicitar a existência de momentos em que o nó *réplica* está indisponível, origem da definição de tarefa de replicação a cargo do *middleware*.

5. Conclusões e Trabalhos Futuros

Neste trabalho de pesquisa foram apresentados resultados de um modelo diferenciado, o qual, diferente de uma proposta anterior que desconsiderava o consumo de tempo quando da realização de operação de replicação, leva em conta a consideração temporal de *stress* de replicação, a qual é necessária para uma melhor representação deste tipo de modelo quando aplicado a *BDDs*.

Com base em tais resultados comparativos entre os modelos com e sem consideração de consumo de tempo de replicação, apresentados com efeitos brandos sobre as possibilidades que tais considerações poderiam causar, concluiu-se que delegar o controle de replicação a um *middleware* lidando com o *Sistema de Gerenciamento de Banco de Dados* (*SGBD*) é o caminho para se evitar sobrecarga de recurso sobre nós replicantes.

Nesta nova representação tem-se agora um modelo discreto, baseado em passos de comunicação entre os nós da arquitetura. Em uma evolução, após a avaliação migrada da abordagem temporal para recursos, este modelo agora não lida mais com *threads* em *loop* com intervalos de verificação, sendo desta maneira um modelo completamente discreto.

Na atual abordagem, o *middleware* tem comportamento similar a um nó comunicante, tornando-se mais simples as verificações necessárias, além de evitar sobrecarga de processamento e armazenamento de fila de operações sobre os *nós de borda*.

Os dados de simulação apresentados demonstram como casos em que os recursos necessários não poderiam ser providos instantaneamente com a consideração real, envolvendo *stress* de replicação, poderiam ser resolvidos ao se utilizar um *middleware* anexo

ao *SGBD* responsabilizado pelo processo de replicação.

Como trabalho futuro espera-se formalizar completamente o modelo de replicação baseado em *nós virtuais*.

Algoritmo 2 Gerenciamento de Operações no Pool de Backup via Middleware

1: $opTempo[inserção|update|deleção] = \{tempo1|tempo2|tempo3\}$
2: $tempo \leftarrow 0$
3: $lista \leftarrow$ nova lista de operações vazia
4: $limite \leftarrow$ tempo limite para atingir cópia completa
5: $tipo \leftarrow$ modo do algoritmo
6: $passo \leftarrow$ passo de incremento do sistema
7: **procedimento** $AdicionaOperação(opNovas)$
8: $i \leftarrow 0$
9: **enquanto** $i < opNovas.tamanho$ **faça**
10: $opNova \leftarrow opNovas.operação(i)$
11: $tempo = tempo + opTempo[opNova.tipo]$
12: $lista.adiciona(opNova)$
13: $i \leftarrow i + 1$
14: **fim enquanto**
15: **se** $tempo \geq limite$ **então**
16: $i \leftarrow 0$
17: **enquanto** $i < réplicas.tamanho$ **faça**
18: $réplica \leftarrow réplicas.acorda(i)$
19: $aloca(réplica)$
20: $j \leftarrow 0$
21: **enquanto** $j < lista.tamanho$ **faça**
22: $op \leftarrow lista.operação(j)$
23: $replica.adicionaOp(op)$
24: $j \leftarrow j + 1$
25: **fim enquanto**
26: $desaloca(réplica)$
27: $i \leftarrow i + 1$
28: **fim enquanto**
29: $tempo \leftarrow 0$
30: $lista \leftarrow$ nova lista vazia
31: **fim se**
32: **se** $\frac{\sum_{i=1}^{\#replicas} replicas[i].c_{livre}}{\#replicas} \leq ANS^{min}_{RASGuarantee}$ **então**
33: chame $RAS_{min}()$
34: **senão se** $c - 2 * ANS^{min}_{RASGuarantee} \geq \frac{\sum_{i=1}^{\#replicas} replicas[i].c_{usado}}{\#replicas}$ **então**
35: chame $RAS_{max}()$
36: **fim se**
37: **fim procedimento**

Referências

Caruana, A. (2002). Service loyalty: The effects of service quality and the mediating role of customer satisfaction. *European Journal of Marketing*, 36(7/8):811–828.

de Souza, R. H. and Dantas, M. A. R. (2015). Mapping QoE through QoS in an approach to DDB architectures: Research analysis and conceptualization. *ACM Computing Surveys (CSUR)*, 48(2).

de Souza, R. H. and Dantas, M. A. R. (2017). An Architectural Recovering Model for Distributed Databases Focused on Resources Availability. Submitted to Distributed and Parallel Databases (DAPD).

de Souza, R. H., Flores, P. A., Dantas, M. A. R., and Siqueira, F. (2016). Architectural recovering model for distributed databases: A reliability, availability and serviceability approach. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 575–580, Messina, Italy.

Gronroos, C. (1984). A service quality model and its market implications. *European Journal of Marketing*, 18(4):36–44.

ISO9241-210 (2010). Iso 9241-210:2010 - human-centred design processes for interactive systems. Available at http://www.iso.org/iso/catalogue_detail.htm?csnumber=21197.

Lewis, R. C. and Booms, B. H. (1983). The marketing aspects of service quality. In Berry, L. L., Shostack, G., and Upah, G., editors, *Emerging Perspectives in Service Marketing*, pages 99–107, Chicago, IL. USA. American Marketing Association.

Parasuraman, A., Zeithaml, V. A., and Bery, L. L. (1985). A conceptual model of service quality and its implication for future research. *Journal of Marketing*, 49:41–50.

Parasuraman, A., Zeithaml, V. A., and Bery, L. L. (1988). Servqual: a multiple-item scale for measuring consumer perceptions of service quality. *Journal of Retailing*, 64(1):12–40.

Parasuraman, A., Zeithaml, V. A., and Bery, L. L. (1994). Alternative scales for measuring service quality: a comparative assessment based on psychometric and diagnostic criteria. *Journal of Retailing*, 70(3):201–30.

Shenker, S., Partridge, C., and Guerin, R. (1997). Rfc 2212: Specification of guaranteed quality of service. Technical report, Internet Engineering Task Force (IETF).

Siewiorek, D. P. and Swarz, R. S. (1998). *Reliable computer systems: design and evaluation*. Natick, Mass. : A K Peters, 3th edition. p. 508.

ur Rehman Laghari, K., Crespi, N., Molina, B., and Palau, C. (2011). Qoe aware service delivery in distributed environment. In *Advanced Information Networking and Applications (AINA), 2011 IEEE Workshops of International Conference on*, pages 837 – 842. Available at http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5763609.

Wolter, K. and van Moorsel, A. (2001). The relationship between quality of service and business metrics: Monitoring, notification and optimization. Technical Report 96, HP Laboratories Technical Report.