Um Ambiente para Testes e Simulações de Protocolos Confiáveis em Sistemas Distribuídos Híbridos e Dinâmicos

Allan Edgard Silva Freitas[1,2], **Raimundo José de Araújo Macêdo**[1]

[1]Laboratório de Sistemas Distribuídos (LaSiD) - Departamento de Ciência da Computação
Universidade Federal da Bahia (UFBA)
Campus de Ondina, Avenida Adhemar de Barros, 40.170-110 - Salvador - BA - Brazil

[2]Instituto Federal da Bahia (IFBA)
Campus de Salvador, Rua Emídio dos Santos, 40.301-015 - Salvador - BA - Brazil

allan@ifba.edu.br, macedo@ufba.br

Abstract. *Distributed systems are usually characterized by a set of processes residing in different sites of a computer network that communicate through message passing. Processes and communication channels are characterized by synchronous or asynchronous timeliness behavior, according to the underlying system (operating system and communication sub-system). Unlike conventional systems, the timeliness characteristics of dynamic and hybrid distributed systems may vary over time, according to the availability of resources and occurrence of failures. Such systems are becoming increasingly common today because of the increasing diversity, heterogeneity and omnipresence of computer networks and devices. Due to its high complexity, such systems are difficult to be tested or verified. In this article, we introduce a new simulation tool for such environments, where several failure models and temporal behavior can be dynamically assigned to processes and communication channels. To illustrate the benefits of such an environment, the performance evaluation of a new group communication protocol is presented.*

Resumo. *Sistemas distribuídos são usualmente caracterizados por um conjunto de processos residentes em sítios variados de uma rede de computadores e que se comunicam através de canais de comunicação. Processos e canais são caracterizados por comportamentos temporais síncronos ou assíncronos a depender dos recursos subjacentes (sistemas operacionais e subsistema de comunicação). Diferentemente dos sistemas convencionais, as características temporais dos sistemas híbridos e dinâmicos variam com o tempo, de acordo com a disponibilidade de recursos e ocorrência de falhas. Tais sistemas estão se tornando cada vez mais comuns nos dias de hoje devido à crescente diversidade, heterogeneidade e onipresença das redes e dispositivos computacionais. Devido à sua grande complexidade, tais sistemas são difíceis de serem testados ou verificados. Neste artigo, introduzimos um novo ambiente de simulação para tais ambientes, onde diversos modelos de falhas e comportamentos temporais podem ser associados dinamicamente a processos e canais de comunicação. Para mostrar o poder de tal ambiente, foi simulado e avaliado o desempenho de um novo protocolo de comunicação em grupo adequado à ambientes híbridos e dinâmicos, onde alguns cenários de execução foram exercitados.*

1. Introdução

Sistemas distribuídos são caracterizados por um conjunto de processos espalhados em diversos computadores de uma rede de comunicação e que se comunicam por troca de mensagens. Uma das principais vantagens dos sistemas distribuídos é a possibilidade de se implementar aplicações tolerantes a falhas, por exemplo, através da replicação de processos, garantindo a continuidade do serviço em execução mesmo que ocorram defeitos em um determinado número de processos e canais de comunicação. Tais sistemas são em geral estudados através de modelos que definem os comportamentos temporais e de ocorrência de falhas dos processos e canais de comunicação [Lynch 1996].

Entretanto, a capacidade de resolver certos problemas de tolerância a falhas em sistemas distribuídos está intimamente ligada à existência de um modelo de sistema adequado. Nesse sentido, há algumas décadas, pesquisadores vêm propondo uma variedade de modelos de resolução de problemas, onde os modelos assíncronos (ou livres de tempo) e síncronos (baseados no tempo) têm dominado o centro das atenções, por serem considerados modelos extremos em termos de resolução de problemas de tolerância a falhas. Por exemplo, o problema de difusão confiável - na presença de canais confiáveis e falhas silenciosas de processos - é solúvel em ambos os modelos [Lynch 1996]. Contudo, o problema de consenso distribuído é solúvel no modelo síncrono, mas não no modelo assíncrono [Fisher et al. 1985]. Por outro lado, serviços de consenso distribuído podem também ser garantidos em ambientes ditos parcialmente síncronos desde que o COMPORTAMENTO SÍNCRONO se estabeleça durante períodos de tempo suficientemente longos para a execução do consenso [Dwork et al. 1988]. Por exemplo, a hipótese *Global Stabilization Time* define um modelo parcialmente síncrono e é necessária para garantir que os protocolos de consenso baseados no detector de defeitos $\diamond\mathcal{S}$ atuem de forma adequada [Chandra and Toueg 1996].

Todos esses modelos de sistema acima são caracterizados por configurações homogêneas e estáticas no que toca os aspectos temporais. Ou seja, uma vez definidas as características temporais dos processos e canais de comunicação, essas não se modificam durante a vida do sistema (estática) e todos os processos e canais de comunicação são definidos com as mesmas características temporais (homogêneo). Uma das exceções no aspecto homogêneo é o sistema TCB [Veríssimo and Casimiro 2002], onde um sistema assíncrono é equipado com componentes síncronas que formam uma *spanning tree* de comunicação síncrona, ou *wormholes*. No entanto, os modelos baseados em *wormholes* são estáticos em relação às mudanças de qualidade de serviços das componentes síncronas.

Para lidar com aspectos dinâmicos e híbridos de modelos de sistemas distribuídos, atendendo às demandas dos novos ambientes com qualidades de serviço variadas, modelos híbridos e dinâmicos foram introduzidos em [Gorender and Macêdo 2002, Gorender et al. 2005, Macêdo 2007, Gorender et al. 2007, Macêdo and Gorender 2009]. Em [Gorender and Macêdo 2002], foi apresentado um algoritmo de consenso que requer uma *spanning tree* síncrona no sistema distribuído, onde processos são síncronos e canais de comunicação podem ser síncronos ou assíncronos. Nos trabalhos [Gorender et al. 2005, Gorender et al. 2007], o requisito de *spanning tree* síncrona foi removido e apresentado soluções para o consenso uniforme em ambientes dinâmicos. Em [Macêdo 2007], o modelo foi generalizado para que processos e canais de comunicação

pudessem variar entre síncrono e assíncrono e foi apresentado um algoritmo de comunicação em grupo capaz de ligar com esses ambientes híbridos e dinâmicos, e finalmente, em [Macêdo and Gorender 2008, Macêdo and Gorender 2009] foi introduzido o modelo híbrido e dinâmico *partitioned synchronous* que requer menos garantias temporais do que o modelo síncrono e onde foi provado ser possível a implementação de detectores perfeitos (mecanismo fundamental para a solução de consenso). Vale salientar que a implementação de detectores perfeitos no modelo *partitioned synchronous* não requer a existência de um *wormhole* síncrono [Veríssimo and Casimiro 2002] ou *spanning tree* síncrona [Gorender and Macêdo 2002], onde seria possível implementar ações síncronas globais em todos os processos, como sincronização interna de relógios. No sistema *partitioned synchronous*, proposto, componentes do ambiente distribuído necessitam ser síncronos, mas os mesmos não precisam estar conectados entre si via canais síncronos, o que torna impossível a execução de ações síncronas distribuídas em todos os processos do sistema - contudo mesmo que processos não estejam em qualquer das componentes síncronas, pode-se tirar proveito das partições síncronas existentes para melhorar a robustez das aplicações de tolerância a falhas.

Apesar das facilidades de modelagem dos sistemas híbridos e dinâmicos, a verificação de certas propriedades dos algoritmos é mais complexa que nos modelos estáticos. Por exemplo, sabe-se que no modelo estático síncrono existe uma solução para o consenso entre n processos para um número máximo f de falhas bizantinas, desde que $n \geq 3f + 1$ [Lamport et al. 1982]. No sistema puramente assíncrono soluções somente existem para $f = 0$. No caso de nosso modelo híbrido e dinâmico, como a qualidade de serviço temporal muda dinamicamente, não se pode prever um valor máximo para f, que pode variar em função do número de componentes síncronos [Gorender et al. 2007].

A análise de desempenho de algoritmos de computação distribuída nas abordagens analítica e de medição [Jain 1991] se torna bastante complexa nos modelos híbridos e dinâmicos, especialmente nos cenários com números grandes de processos e canais de comunicação, cujas configurações variam com o tempo. Nesses casos, o uso de simulação torna-se uma alternativa viável. No entanto, percebe-se que nos simuladores existentes para sistemas distribuídos, a exemplo do Neko [Urban et al. 2001], não se representam de forma adequada as restrições inerentes ao nosso modelo híbrido e dinâmico. O desenvolvimento de simuladores para tais modelos é, portanto, um desafio a ser enfrentado pela comunidade de pesquisa em simulação de sistemas distribuídos.

O presente trabalho responde a esse desafio, apresentando um novo simulador adequado a estes cenários acima descritos. Além disso, procuramos o desenvolvimento de um simulador que permitisse a descrição dos protocolos através de linguagens de programação convencionais, de modo a reutilizar o código da simulação em aplicações reais. Para validar o simulador proposto, foi analisado o comportamento de um protocolo de comunicação em grupo voltado para o modelo híbrido e dinâmico e um outro protocolo convencional para sistemas síncronos [Cristian et al. 1986, Cristian et al. 1988, Cristian et al. 1995]. Nossa análise permitiu não somente validar a adequação do simulador, mas também permitiu aferir as vantagens do nosso protocolo de comunicação em grupo em relação ao protocolo de Flaviu Cristian et al. em cenários de execução específicos (quando o sistema se comporta de forma síncrona). Além dessas simulações, incluímos uma outra simulação de um detector perfeito sobre o modelo híbrido e dinâmico

descrito em [Macêdo and Gorender 2009]. Portanto, as principais contribuições deste artigo são: um novo simulador para sistemas distribuídos híbridos e dinâmicos e também a avaliação de desempenho, realizada pela primeira vez, do protocolo de comunicação em grupo proposto em [Macêdo 2007].

O artigo organiza-se da seguinte forma: a seção 2 apresenta uma discussão sobre trabalhos correlatos em simulação; a seção 3 detalha a estrutura do ambiente de testes e simulações proposto; a seção 4 apresenta a utilização do simulador proposto, através da simulação e avaliação de um protocolo de comunicação em grupo genérico em diferentes cenários; por fim, a seção 5 mostra as considerações finais.

2. Trabalhos Correlatos

Um simulador de sistemas distribuídos deve permitir expressar de modo mais adequado possível o ambiente desejado, bem como prover um conjunto de ferramentas adequadas à reutilização e rápida prototipação de novos algoritmos, permitindo representar a infra-estrutura, suas mudanças, os processos e seus canais, bem como o progresso da computação distribuída. Existe uma lista extensa de proposta de simuladores, que procuram se especializar em cenários distintos, como redes sem fio, grades computacionais, entre outros. Uma taxonomia e discussão abrangente sobre tais simuladores não cabe no escopo deste artigo e pode ser encontrada em [Sulistio et al. 2004]. Abaixo listamos alguns mais significativos.

O simulador Neko [Urban et al. 2001] - baseado no *framework* de simulação de eventos discretos *SimJava* [Howell and McNab 1998] - é amplamente utilizado pela comunidade de sistemas distribuídos. O Neko permite combinar simulação e medição, formando um *framework* único baseado em Java, o qual pode rodar em modo simulado ou ser executado como aplicação de rede. A grande vantagem em utilizar o Neko é, portanto, a reutilização do código de simulação em uma aplicação real. Contudo, a forma de composição dos relógios dos processos simulados no Neko apresenta limitações para a simulação de tarefas de tempo real crítico - pois suas primitivas de agendamento de tarefas não garantem a execução em uma janela de tempo estrita. Ainda, o Neko não permite variar a topologia de conexões, ao longo da simulação. Essas restrições inviabilizaram a utilização do Neko em nosso modelo híbrido e dinâmico.

Parsec [Bagrodia et al. 1998] é um simulador de modelos de rede complexos de larga escala. Tal ferramenta utiliza uma linguagem de simulação própria dita Parsec (dito *language-based* conforme [Sulistio et al. 2004]), o que limita o reuso em aplicações reais.

Um ambiente bastante popular para a comunidade de redes de computadores é o NS-2 [Breslau et al. 2000], que é um simulador de eventos discretos baseado em C++ e *Object Tcl*. O NS-2 provê simulação de protocolos, os quais acessam topologias pré-definidas a partir de *scripts oTcl*.

Como a simulação em sistemas distribuídos apenas modela o comportamento fim-a-fim de canais, a simulação desses pode ser simplificada através de comportamento probabilístico nos tempos de entrega de mensagens e perda de pacotes, não importando o comportamento concreto dos protocolos das camadas subjacentes (rede, enlace e física). Contudo, é desejável compor cenários dinâmicos de infra-estrutura, bem como aplicações complexas que utilizem os serviços propostos pelos protocolos, tornando as simulações

mais realísticas. Neste contexto, é desejável reproduzir tempo real, simulando relógios físicos, bem como permitindo o agendamento de tarefas e a reprodução de eventos em janelas temporais estritas - o que não é possível em ambientes como o Neko, como afirmamos acima. O simulador proposto neste artigo atende a tais desafios.

3. Ambiente para Testes e Simulações em Sistemas Distribuídos Híbridos e Dinâmicos

3.1. Modelo de Sistema Distribuído Híbrido e Dinâmico

Um sistema distribuído é formado por um conjunto Π de processos $p_1, p_2, ..., p_n$, e um conjunto χ de canais de comunicação $c_1, c_2, ..., c_m$. Cada canal c_i de χ liga somente dois processos distintos de Π e cada processo de Π está ligado a todos os outros processos de Π. Portanto, nosso sistema forma um grafo simples completo não direcionado $DS(\Pi, \chi)$ com $(n \times (n-1))/2$ arestas. Tanto processos em Π quanto canais em χ podem ser *timely* ou *untimely*. Um dado processo é *timely* se existe valor máximo conhecido (digamos, ϕ) para a execução de passos de computação em p_i. Da mesma forma, um canal c_i é *timely* se uma mensagem é transmitida em c_i dentro de um limite de tempo limitado e conhecido, digamos δ. Do contrário, processos e canais são *untimely*. δ e ϕ são parâmetros do sistema de execução e garantidos por mecanismos de sistemas operacionais e redes de tempo real. Assumimos também que processos e canais podem mudar sua qualidade de serviço de forma que possam alternar entre *timely* e *untimely*. Além disso, é possível, a partir do ambiente, identificar localmente qual a qualidade de serviço de um processo ou canal. Cada processo p_i possui acesso a um relógio local $CLOCK_i$, o qual expressa o tempo mensurado pelo temporizador local. A taxa de ajuste deste temporizador local pode derivar em relação ao tempo real dentro de um limite máximo ρ, devido, por exemplo, a imprecisão dos cristais.

Um sistema distribuído é híbrido se seus elementos, processos e canais, podem apresentar comportamentos distintos (i.e. um conjunto de processos e canais com comportamento *timely* ou síncrono; e outro *untimely* ou assíncrono). O sistema é dito dinâmico, se este comportamento pode variar ao longo do tempo (i.e. um processo ou canal com um comportamento síncrono pode alternar para um comportamento assíncrono, ou vice-versa).

Um sistema distribuído pode apresentar falhas de funcionamento, ou seja apresentar comportamento diverso a funcionalidade esperada. Estas falhas podem ocorrer tanto em processos quanto em canais. O modelo de falhas do sistema é tão essencial quanto a especificação do comportamento normal, uma vez que os mecanismos que permitam tolerar falhas e manter o comportamento esperado dependem de que tipo de falhas ocorrem. Uma vez conhecidas com um dado grau de probabilidade as falhas que possam ocorrer, determinam-se quais falhas compõem o modelo de falhas apropriado [Cristian 1991].

3.2. Simulador Proposto

No desenvolvimento de um ambiente apropriado de simulação de protocolos em sistemas distribuídos híbridos e dinâmicos, é desejável reproduzir comportamentos existentes no mundo real: a variação da topologia do sistema, comportamento de tempo real crítico e mudanças de comportamento em processos e canais. Isto é obtido através da definição do

comportamento assumido por processos e canais, bem como a dinâmica deste comportamento e da infra-estrutura subjacente (que pode implicar na renegociação de canais entre processos), bem como o modelo de falhas de canais e de processos.

Desta forma, é proposto um simulador composto por um pacote de classes Java, o qual permite definir um ambiente e os processos, negociar canais entre os processos e reproduzir modelos de falhas de processos e canais existentes na literatura, bem como, ao longo da simulação alterar este ambiente. Por exemplo, pode-se definir processos síncronos ou assíncronos e respectivas distribuições de probabilidade para a ocorrência de falhas, dentro de um modelo de falhas previamente definido para os processos (modelo *crash*, por exemplo). O mesmo pode ser feito em relação ao canais.

Conforme a taxonomia apresentada em [Sulistio et al. 2004], pode-se afirmar que nosso simulador tem um motor baseado em execução serial de eventos discretos, orientado pelos eventos e pelo tempo (tarefas podem ser agendadas pelo tempo ou na ocorrência de eventos). A ferramenta é dita *library-based*, pois os processos de fato utilizam um pacote Java que provê a troca de mensagens no ambiente de simulação, o que poderia ser substituído, por exemplo, por *sockets* em um ambiente real.

Nosso ambiente de simulação, denominado **HDDSS** (em referência a "'hybrid and dynamic distributed system simulator"' - simulador de sistemas distribuídos híbridos e dinâmicos), possui dois modos de operação no que concerne aos relógios. Tarefas podem ser agendadas em cada processo p_i para um instante t dado pelo relógio local ao processo ($CLOCK_i$). No primeiro modo, cada processo possui seus próprios temporizador e relógio, com a atualização dos temporizadores em relação a um tempo global virtual guardando a taxa máxima de desvio ρ - permitindo tarefas de tempo real crítica e protocolos de sincronização de relógios. No último, cada processo possui seu próprio relógio sem simulação do *drift-rate*, embora seja possível manter um intervalo máximo de ϵ, se necessário - simulando-se apenas o efeito da sincronização de relógios.

A classe **Simulador** implementa os dois modos de operação, e um conjunto de processos e de canais, cujo comportamento genérico é definido pelas classes **Agente** e **Canal**. Um processo que implementa um comportamento específico de um dado protocolo é uma classe herdada de **Agente**, e da mesma forma, um comportamento específico de canal de comunicação (por exemplo, tempo de entrega determinística ou por uma distribuição Poisson) é implementado por herança da classe **Canal**. Além disto, de modo similar ao Neko, a comunicação entre processos se dá por invocação de métodos do simulador para troca de mensagens, no qual a abstração de canais é simulada, considerando o ambiente definido. Os processos invocam métodos de envio de mensagens, recepção das mensagens no *buffer* de recepção e podem agendar a entrega de mensagens recebidas, bem como processar a entrega realizando ações específicas. Topologias arbitrárias são criadas no método de início de simulação e podem ser alteradas ao longo da evolução do tempo global de simulação.

A implementação do modelo de falhas independe do código de protocolo criado. É importante observar que cada elemento do conjunto de processos ou do conjunto de canais, pode ser uma instância de uma classe distinta de agente ou de canal, respectivamente, implementando protocolos distintos que interagem entre si, ou canais de comunicação com comportamentos distintos. De modo similar, cada um destes ele-

mentos pode implementar um modelo de falhas distinto (ou nenhum, não apresentando falhas). Ao implementar um modelo de falhas se define a probabilidade de ocorrência das falhas. Ainda, o simulador permite que, ao longo do tempo, alterar um canal entre dois processos, definindo um novo comportamento, instanciando uma outra classe. Uma classe com métodos probabilísticos distintos (i.e. uniforme, log-normal etc.) foi criada para dar suporte a construção das classes.

A simulação é executada através da classe **Simulador**, a partir de um arquivo texto inicial de configuração, o qual define a quantidade de agentes, o tipo de cada agente, o comportamento dos canais, os modelos de falhas dos agentes e dos canais, bem como a especificação de uma dinâmica para o sistema em tempo de simulação (e.g. a forma como canais podem alterar seu comportamento ao longo da simulação). A saída da simulação é dada em formato de texto, com informações estatísticas acerca de métricas previamente configuradas (por exemplo, tempo médio e desvio padrão de entrega de mensagens) e com detalhes da ocorrência de eventos, também previamente definidos, como, por exemplo, a recepção de mensagens pelos agentes.

Por restrições de espaço não descreveremos neste artigo toda a hierarquia de classes do simulador.

4. Simulação e Avaliação de Protocolos para Sistemas Híbridos e Dinâmicos

Nesta sessão apresentaremos a avaliação de um protocolo de comunicação em grupo em cenários distintos. Para esse fim, utilizaremos o protocolo Genérico Adaptativo de Comunicação em Grupo, inicialmente descrito em [Macêdo 2007], tendo sido suas propriedades formalizadas e detalhadas em [Macêdo 2008]. O presente artigo apresenta uma avaliação preliminar desse protocolo.

4.1. Um Protocolo Genérico Adaptativo de Comunicação em Grupo

O *framework Causal Blocks* [Macedo et al. 1995] é um protocolo de comunicação em grupo utilizado em ambientes sem requisitos temporais que permite comunicação em grupo com ordenação causal ou total. Cada processo p_i mantém um contador de blocos BC_i, iniciado com um valor não negativo. Mensagens são transmitidas por p_i com BC_i como rótulo temporal $m.b$, respeitando-se causalidade em potencial. O contador BC_i avança de um a cada mensagem transmitida por p_i; e na entrega de uma mensagem m em p_i, definimos $BC_i = maxBC_i, m.b$. Isto garante que para mensagens distintas $m, m' : envio(m) \rightarrow envio(m') \Rightarrow m.b < m'.b$. Cada processo mantém uma matriz de blocos, onde cada bloco $B[b]$ indica o envio/recebimento de mensagens de *timestamp* lógico b por cada um dos processos P_i, através de uma marcação '+' na célula $B[m.b, i]$, onde $m.b$ é o *timestamp* de uma mensagem m. Um bloco é dito completo se $\forall i, BM[b, i] =' +' \vee \exists b' > b | BM[b', i] =' +'$. Por exemplo, a tabela 1 representa os blocos causais mantidos por P_1: por definição, o bloco 2 está completo devido a mensagem de rótulo 2 de P_2 e às mensagens de rótulo 3 de P_1 e P_3 - ainda, deve-se observar que se um bloco está completo, os blocos anteriores também o estão (i.e. bloco 1), em face da propriedade FIFO dos canais.

A entrega de mensagens se dá através da completude de blocos. De modo que a garantir a entrega continua de mensagens, é previsto um mecanismo denominado *time-silence* em que um processo força a completude de um bloco $m.b$ criado no instante t em

Tabela 1. exemplo de blocos causais de processo P_1

	P_1	P_2	P_3
1	+		
2		+	
3	+		+

até $t + ts$, enviando uma mensagem com $m.b$ caso o protocolo de aplicação se mantenha inativo. Isto garante que todos blocos serão entregues, mesmo quando há inatividade da aplicação.

Na presença de um ambiente (processos e canais) síncrono, é possível inferir os limites temporais para a completude de blocos após sua criação em um processo p_i:

- TC1: $(t_i + ts(m.b) + 2\Delta_{max})(1 + \rho)$, se a mensagem m que criou o bloco foi enviada por p_i
- TC2: $(t_i + ts(m.b) + 2\Delta_{max} - \Delta_{min})(1 + \rho)$, se a mensagem m que criou o bloco [enviada de um processo síncrono e com comunicação síncrona com p_i] foi recebida por p_i

onde Δ_{max} e Δ_{min} são, respectivamente, os limites temporais máximo e mínimo da transmissão da mensagem através do canal de comunicação $\chi(i, j)$.

Em um ambiente híbrido e dinâmico, no qual a infra-estrutura possa mudar ao longo do tempo, de modo que processos e canais possam alternar sua qualidade de serviço (i.e. síncrono e assíncrono) - o que pode ocorrer por renegociação ou por falhas, pode-se equipar a infra-estrutura subjacente com um mecanismo de monitoramento. Este mecanismo provê a cada processo a informação adequada de qualidade de serviço de um dado canal ou processo (a mudança de estado pode ser informada dentro de algum atraso temporal de modo que processos distintos não necessitam ter a mesma visão da qualidade de serviço relacionada a um terceiro processo ou a um canal em um dado instante de tempo). Modificações na dinâmica da qualidade de serviço e ocorrência de colapso (*crash*) de processos, permitem a cada processo P_i observar os subconjuntos de Π: $live_i$, $uncertain_i$ e $down_i$, como definidos em [Gorender et al. 2007]. Se $p_j \in live_i$, p_j é síncrono e está conectado a (ao menos) um outro processo síncrono p_k (não necessariamente $k = i$) por um canal bidirecional síncrono (p_i, p_k). Caso contrário, $p_j \in uncertain_i$. Se o processo que colapsa estava em $live_i$, então ele é movido de $live_i$ para $down_i$. Estes conjuntos são dinamicamente atualizados pelos mecanismos de monitoramento ora descritos, e processos podem falhar somente por travamento da execução (i.e. *crashing*). Falhas bizantinas não são toleradas e canais são confiáveis e FIFO.

Desta forma, o mecanismo de monitoramento subsidia a composição de um protocolo genérico e adaptativo para comunicação em grupo, o qual utiliza do conhecimento de limites temporais quando percebe sincronia no ambiente. Este protocolo, com o conhecimento dos limites temporais $TC1$ e $TC2$ estende o mecanismo do *Causal Blocks* para *Timed Causal Blocks* [Macêdo 2007], dito **TimedCB**. No **TimedCB**, caso o sistema seja puramente síncrono, a completude dos blocos rege-se pelas regras TC1 e TC2; caso apenas alguns canais e processos sejam síncronos, as colunas correspondentes regem-se pelas regras TC1 e TC2, e as demais colunas do bloco podem se completar sem atender a limites temporais.

O protocolo é denominado genérico adaptativo. O mesmo é genérico pelo fato de que pode trabalhar tanto de forma síncrona quanto assíncrona, dependendo somente da sincronia atual da camada de execução subjacente. Ainda, o protocolo se ajusta a um sistema que é híbrido, então o protocolo é adaptativo no sentido que se adaptará ao ambiente disponível em tempo de execução, por meio do conhecimento obtido através do mecanismo de monitoração subjacente.

O protocolo genérico adaptativo se baseia em visões de grupo $v_i \subseteq \Pi$ (estabelecidas pela detecção de ingressos, saídas e falhas de processos do grupo), em que uma visão representa o conjunto de membros do grupo que são mutuamente considerados operacionais em um dado instante da existência do grupo. Cada vez que uma mudança ocorre na visão do grupo, uma nova visão é instalada por um protocolo de *group membership*. Cada visão instalada por um processo é associada com um número que incrementa monotonicamente com a instalação das visões, $v_i^k(g)$ denota a visão de número k instalada por p_i. Na criação do grupo, todos os membros instalam a mesma visão inicial $v_i^1(g) = \Pi$. Quaisquer modificações subsequentes na configuração do grupo resultam em novas visões sendo instaladas, formando a sequência $v_i^1, v_i^2, ..., v_i^k$, onde k representa um dado momento na história de visões. Um processo p_i somente difunde mensagens para sua visão atual. Ainda, o mesmo deve possibilitar entrega de mensagens de acordo com as propriedades de *agreement*, *causal order* e *uniform total order*, possibilitando por exemplo a replicação ativa de servidores. O *membership* dinâmico é requerido, por exemplo, para manter um dado número de réplicas funcionais ativas - e a propriedade de *failure detection* garante que ao longo da evolução das visões, falhas por *crash* serão detectadas e processos falhos exclusos, bem como a propriedade de *unique sequence of views* estabelece que todos processo instalam a mesma sequencia de visões, concordando sobre as mesmas - propriedades adicionais asseguram que processos só instalam as visões se pertencerem as mesmas e, caso sejam retirados de uma sequência de visões foi por terem falho por *crash* ou serem suspeitos de tal. Por limitações de espaço, não faremos a prova e discussão detalhada de tais propriedades.

Este protocolo tem como base o mecanismo de **TimedCB**, utilizando em caso de falhas (percebidas pelo não atendimento dos limites temporais de completude $TC1$ e $TC2$) de um mecanismo de consenso para concordar no conjunto de mensagens e de quais processos comporão a próxima visão. O consenso utilizado é o adaptativo [Gorender et al. 2007], de modo a apresentar comportamento otimizado para um ambiente híbrido. Por simplicidade, o protocolo genérico adaptativo pode ser denominado pelo seu mecanismo base, o **TimedCB**.

As propriedades especificadas para o protocolo requerem que todos processos corretos do grupo entreguem o mesmo conjunto de mensagens na mesma ordem. Para atender a tais propriedades, um dado processo membro p_i deve verificar as condições de entrega *safe1*, uma mensagem m recebida é entregável se $BM[m.b]$ é completo, e *safe2*, mensagens são entregues em ordem não decrescente de seus números de bloco - uma ordem pré-determinada de entrega é imposta sobre mensagens entregáveis de um mesmo número de blocos.

O algoritmo 1 descreve os passos executados cada vez que uma nova mensagem é enviada ou recebida. Depois de criar o bloco causal relacionado (se não existir) e definir um *timeout* para sua completude, a mensagem é armazenada no *buffer* local. De-

pois disto, a tarefa de entrega (algoritmo 2) será sinalizada em ordem para verificar as condições de entregas de todos blocos causais incompletos (incluindo os recentemente criados) e as duas condições de segurança são verificadas garantindo a ordenação total (respeitando a causalidade). Sem falhas, o *time-silence* garante a completude dos blocos. Suponha que um processo p_k falhe por crash e, consequentemente, o *timeout* da completude de um bloco ($TC1$ ou $TC2$) expira e p_i para um $BM[m.b]$. Em ordem para proceder com entrega de mensagens, um novo membership para g é estabelecido excluindo p_k. E de modo a garantir que todos membros terão a mesma visão, uma primitiva de difusão confiável, $rmcast(ChangeViewRequest, B), B = m.b$, é aplicada para disparar a mudança de visão (algoritmo 3). A propriedade de concordância da difusão confiável garante que se um dos membros operacionais do grupo entregam a mensagem $(ChangeViewRequest, B)$, todos os demais o fazem. As requisições são então processadas pela tarefa de mudança de visão, o qual executa o consenso apresentado em [Gorender et al. 2007] para entregar os blocos instáveis, conforme demostrado em [Macêdo 2008].

Algoritmo 1: Executado por um processo p_i em um evento de envio/recepção de uma mensagem m

```
se BM[m.b] não existe então
    cria BM[m.b];
    se p_i = m.remetente então
        defina timeout TC1 para BM[m.b];
    senão
        defina timeout TC2 para BM[m.b];
    fim
fim
armazene m em buffer local;
sinalize para o algoritmo 2;
```

Algoritmo 2: Tarefa de entrega de mensagens

```
se algum bloco causal é completo então
    entregue mensagens de acordo com safe1 e safe2;
    cancele timeouts para blocos causais completos;
fim
```

Algoritmo 3: Tarefa executada por p_i na expiração de um timeout para $BM[B]$

```
rmcast(ChangeViewRequest, B);
```

4.2. Simulações

A seguir são apresentados 3 cenários de testes e simulações. O primeiro avalia o desempenho do protocolo genérico adaptativo proposto em um cenário síncrono, comparado ao desempenho do conjunto de protocolos descritos em [Cristian et al. 1988, Cristian et al. 1995], que permitem comunicação em grupo com *membership* e entrega atômica em um ambiente síncrono com o auxílio de sincronização de relógios. Por simplicidade, tais protocolos acima serão referenciados pelo protocolo de *membership*, *Periodic*

Group Creator, dito **PGC**; bem como o protocolo genérico adaptativo será referenciado por **TimedCB**.

No **PGC** cada processo envia mensagens de verificação de membership a cada período π, o algoritmo de entrega atômica baseia-se em inundação, entrega agendada a partir dos relógios físicos, considerando o atraso máximo da rede, retransmissões e uma diferença máxima de ϵ entre os relógios mantida por um mecanismo de sincronização. Em ausência de mensagens da aplicação, o **TimedCB** usa o período ts do *time-silence* para completude dos blocos, o que permite a verificação de membership, de forma similar, ao **PGC**. Assim, em nossas simulações, consideramos cenários em que $ts = \pi$. Ainda, no **TimedCB** não é necessária a sincronização de relógios físicos.

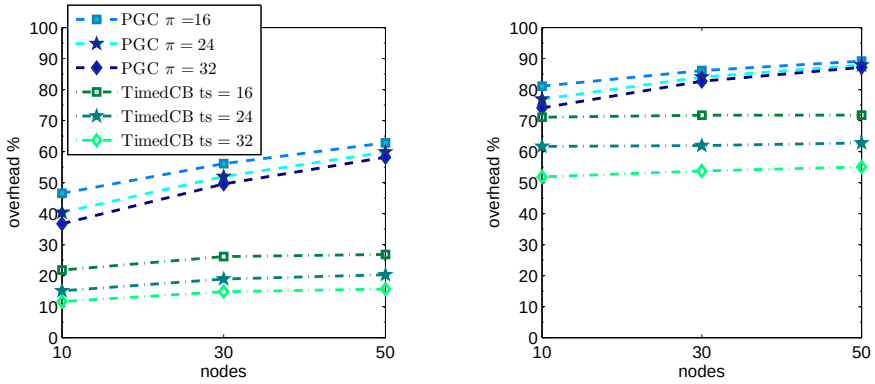
O objetivo é avaliar se o protocolo desenvolvido para ambientes híbridos possui desempenho otimizado em ambientes síncronos sem falhas. Como parâmetros de simulação, temos canais de comunicação determinísticos com $\delta_{MAX} = 14$ unidades de tempo e $\delta_{MIN} = 10$. A topologia da rede é *full-connected*. O algoritmo de sincronização utilizado pelo **PGC** é ajustado para manter os relógios dos processos dentro de um intervalo de $\epsilon = 4$ unidades de tempo. Os fatores da simulação são os nós (10, 30 ou 50) e os períodos ts e π (16, 24 ou 36).

Cada processo gera ou não uma mensagem de aplicação a cada avanço de uma unidade do tempo, de acordo com uma distribuição de Bernoulli. Cada simulação foi feita em 5 replicações do experimento e uma janela de 500 unidades de tempo. Dois perfis de carga são simulados: **A** e **B**. No cenário **A** ($P = 0,1$), são gerados em média 493, 1433 e 2379 mensagens de difusão da aplicação, respectivamente, para 10, 30 e 50 nós. No cenário B ($P = 0,02$), são geradas, respectivamente, em média 109, 302 e 494 mensagens de difusão da aplicação. Em ambos cenários, foram mensurados o percentual de mensagens de protocolo em comparação com o total de mensagens geradas (incluindo-se as de aplicação), e o atraso de entrega de mensagens.

Com o aumento da carga, verifica-se, que, como o **TimedCB** faz proveito das mensagens da aplicação, apresenta *overhead* menor que o conjunto de protocolos de Cristian, em especial pelo **PGC** sempre gerar periodicamente mensagens de protocolo para manter o membership. Define-se *overhead* o percentual de mensagens próprias do protocolo em comparação ao total de mensagens. A figura 1(a) apresenta o percentual de *overhead* pelo número de processos para execuções dos protocolos sob períodos distintos de ts e π. Vê-se que o aumento do período de verificação $ts = \pi$, reduz de forma significativa o *overhead* devido as mensagens dos protocolos. Observe que mesmo no cenário B [figura 1(b)], de baixa carga, o TimedCB apresenta menor overhead que o **PGC**.

No cenário **B**, de menor carga, espera-se que o mecanismo de *time-silence* do **TimedCB** atue de forma mais efetiva, dado a inatividade. Assim, observa-se que tal aumento de ts têm maior impacto no atraso da entrega das mensagens no caso do protocolo genérico. Por exemplo, para 50 nós, a variação de ts de 16 à 32 implica na variação de $24,307 \pm 6,584$ para $40,182 \pm 9,727$ no atraso de entrega. No cenário **A**, tal variação é de $19,799 \pm 6,079$ para $34,161 \pm 9,023$.

O segundo cenário é dinâmico, com 50 nós, avaliando-se o protocolo **TimedCB** com $ts = 32$. Neste cenário, a qualidade de serviço dos canais degrada de entrega determinística para um atraso probabilístico, ao longo da janela de execução. Neste caso,



(a) cenário A (b) cenário B

Figura 1. Gráficos de Simulação de PGC e TimedCB em ambiente síncrono

verifica-se um leve aumento no atraso médio da entrega de mensagens de $32,61 \pm 9,4$ no ambiente síncrono para $32,85 \pm 9,45$ no cenário dinâmico. O algoritmo adaptou a entrega de mensagens, de acordo com a qualidade de serviço dos canais, e verificou-se a flexibilidade do simulador para compor cenários dinâmicos.

Por fim, foi simulado um ambiente com características híbridas, composto por canais síncronos e assíncronos e processos síncronos. Este cenário é ilustrado na figura 2, e compõe-se por um sistema não-síncrono, no qual cada processo está em uma partição síncrona com ao menos outro processo e é possível compor um arranjo de detectores perfeitos $\mathcal{P}$ para todo o sistema distribuído, como discutido em [Macêdo and Gorender 2009]. Neste último cenário, a comunicação em grupo ocorre por meio do protocolo genérico adaptativo, sendo que os processos podem falhar por *crash*.

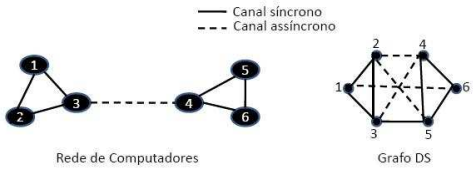


Figura 2. cenário híbrido

O protocolo **TimedCB** é modificado para considerar *timeouts* somente para os canais síncronos. O uso do mecanismo de *time-silence* e destes *timeouts* implementa o comportamento de um detector perfeito $\mathcal{P}$: ao ocorrer tal evento, a mudança de visão é requisitada a todos, conforme os algoritmos apresentados. O cenário simulado é composto por 6 processos, canais síncronos com $\delta_{MAX} = 14$ unidades de tempo e $\delta_{MIN} = 10$ e canais assíncronos baseados em um atraso de $\Delta_{min} = 10$ adicionado a uma distribuição exponencial com valor médio de 14. Neste caso, para 5 replicações de cada experimento, verificou-se que a detecção da falha ocorre em média 45, 49 e 56 unidades de tempo da mesma, respectivamente para $ts = 16$, 24 e 32 - sendo então difundida para todo o grupo.

Assim, o primeiro cenário simulou um cenário homogêneo (síncrono); no segundo cenário, o ambiente é dinâmico (degradação de *QoS*), e, por fim, um ambiente híbrido (composto por canais síncronos e assíncronos) é composto.

5. Considerações Finais

Sistemas distribuídos híbridos e dinâmicos variam com o tempo, de acordo com a disponibilidade de recursos e ocorrência de falhas. Para verificação e testes de protocolos adequados a estes ambientes pode-se usar simulação, considerando o tratamento de requisitos temporais críticos, se existentes, bem como a capacidade de expressar processos e canais com comportamentos heterogêneos, reflexo da infra-estrutura subjacente. Ainda, é necessário simular mudanças no ambiente e permitir renegociar de canais, bem como poder empregar diferentes modelos de falhas, se preciso.

Para avaliação de desempenho de protocolos distribuídos em tais ambientes, foi apresentado neste artigo um novo simulador, onde diversos modelos de falhas e comportamentos temporais podem ser associados dinamicamente a processos e canais de comunicação. Demonstramos o poder de tal ambiente, na simulação e avaliação do desempenho de um novo protocolo de comunicação em grupo adequado a ambientes híbridos e dinâmicos, onde distintos cenários de execução foram exercitados. Também foi simulado um protocolo clássico de comunicação em grupo e comparado seu desempenho com o protocolo genérico-adaptativo por nós proposto. Além disso, simulamos o uso de detectores perfeitos em um ambiente híbrido. As simulações permitiram validar a adequação do simulador e também destacar as vantagens do protocolo genérico-adaptativo em alguns dos cenários. Como trabalho futuro, introduziremos características de mobilidade e disponibilizaremos o código para a comunidade.

Referências

Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., and Song, H. (1998). Parsec: A Parallel Simulation Environment for Complex Systems. *Computer*, pages 77–85.

Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varadhan, K., Xu, Y., et al. (2000). Advances in Network Simulation. *Computer*, pages 59–67.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78.

Cristian, F., Aghili, H., and Strong, H. (1986). Approximate clock synchronization despite omission and performance faults and processor joins. In *Proc of the 16th Int Symp on Fault-Tolerant Computing*.

Cristian, F., Aghili, H., Strong, R., and Volev, D. (1995). Atomic Broadcast: from simple message diffusion to byzantine agreement. In *Proc of the 25th Int Symp on Fault-Tolerant Computing*.

Cristian, F., Center, I., and San Jose, C. (1988). Agreeing on who is present and who is absent in a synchronousdistributed system. In *Digest of Papers of the 18th Int Symp on Fault-Tolerant Computing (FTCS-18)*, pages 206–211.

Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

Fisher, M. J., Lynch, N., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Gorender, S., de Araújo Macêdo, R., and Raynal, M. (2007). An Adaptive Programming Model for Fault-Tolerant Distributed Computing. *IEEE Transactions on Dependable and Secure Computing*, pages 18–31.

Gorender, S. and Macêdo, R. J. A. (2002). Um modelo para tolerância a falhas em sistemas distribuídos com qos. In *Anais do XX Simpósio Brasileiro de Redes de Computadores (SBRC)*, pages 277–292.

Gorender, S., Macêdo, R. J. A., and Raynal, M. (2005). A qos-based adaptive model for fault-tolerant distributed computing (with an application to consensus). In *Proc of IEEE/IFIP Int Conf on Computer Systemas and Networks (DNS05)*, pages 412–421.

Howell, F. and McNab, R. (1998). SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling. In *Proc of the 1st Int Conf on Web-based Modelling and Simulation*.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. Wiley e Sons.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc.

Macêdo, R. J. A. (2008). Adaptive and dependable group communication. Technical Report 001/2008, Distributed Systems Laboratory (LaSiD) - Federal University of Bahia, Salvador, Brazil.

Macêdo, R. J. A. and Gorender, S. (2008). Detectores perfeitos em sistemas distribuídos não síncronos. In *Anais do IX Workshop de Testes e Tolerância a Falhas (WTF)*.

Macedo, R., Ezhilchelvan, P., and Shrivastava, S. (1995). Newtop: A Fault-Tolerant Group communication Protocol. In *Proc of the 15th IEEE CS Int Conf on Distributed Computing Systems*, pages 296–306.

Macêdo, R. J. A. (2007). An Integrated Group Communication Infrastructure for Hybrid Real-Time Distributed Systems. In *Proc of the 9th Workshop on Real-Time Systems*, pages 81–88.

Macêdo, R. J. A. and Gorender, S. (2009). Perfect Failure Detection in Non-synchronous Distributed Systems. In *Proc of the 4th Int Conf on Availability, Reliability and Security, Fukuoka, Japan, March/2009, IEEE CS Press*.

Sulistio, A., Yeo, C., and Buyya, R. (2004). A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software- Practice and Experience*, 34(7):653–673.

Urban, P., Defago, X., and Schiper, A. (2001). Neko: a single environment to simulate and prototype distributed algorithms. In *Proc of 15th Int Conf on Information Networking*, pages 503–511.

Veríssimo, P. and Casimiro, A. (2002). The timely computing base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930.