Detecção de Estado Global em Sistema de Computação Móvel

Dário Vieira Conceição
Ricardo O. Anido[1]
Instituto de Computação - IC
Universidade Estadual de Campinas - UNICAMP
Caixa Postal 6176 - CEP 13083-970
Campinas - São Paulo - Brasil
email: {973928, ranido}@dcc.unicamp.br

Resumo

A obtenção do estado global de aplicações distribuídas em ambientes de computação móvel é dificultada devido a um conjunto de restrições e características deste ambiente: alta latência, baixa largura de banda, restrição no consumo de energia, baixa capacidade de armazenamento, falta de um local confiável para armazenamento nas unidades móveis, mudança de localização e frequentes desconexões. Nesse artigo apresentamos um novo protocolo para obtenção de estado global. O protocolo utiliza a técnica desenvolvida por [JJ94] para reduzir o número de informações que são enviadas em uma mensagem. Levando-se em consideração as características e restrições dos sistemas móveis, essa redução é de vital importância para o ambiente de computação móvel.

Palavras chaves: computação móvel, obtenção de estado global, gravação de estado local, comunicação sem fio, desconexão, sistemas distribuídos.

Abstract

Checkpointing in mobile applications is more complex than in static distributed systems due to a set of characteristics of the mobile environment: high latency, low bandwidth, frequent disconnections, and severe constraints on power consumption and disk usage. In this paper we present a new protocol for checkpointing in mobile computer systems. The proposed protocol exchanges less information during the checkpoint process than previously known protocols. Considering the characteristics of mobile systems, the reduction obtained in network bandwidth is of primary value for mobile applications.

Keywords: checkpointing, mobile computing, wireless communication, disconnection, distribuited systems.

1 Introdução

Computadores móveis e dispositivos como PDAs (Personal Digit Assistants) serão usados crescentemente em aplicações que requerem a interação entre várias pessoas. Mecanismos de obtenção de estado global são necessários para garantir que apesar da desconexão, falha ou perda de um ou mais desses dispositivos, a computação executada pelo grupo possa ser

[1]Trabalho parcialmente financiado pelo CNPq e FAPESP

restaurada a partir do último estado global salvo. Obtenção de estado global é um problema bastante estudado no caso de sistemas distribuídos, mas em computação móvel o problema é dificultado devido a um conjunto de restrições e características intrínsecas ao ambiente móvel.

Dois protocolos para obtenção de estado global em sistemas móveis foram propostos recentemente [AB94], [NF97]. Nesse artigo apresentamos um novo protocolo para obtenção de estado global, que utiliza a técnica desenvolvida em [JJ94] para reduzir o tamanho das mensagens. Levando-se em consideração as restrições de força, banda passante e latência nos sistemas móveis, essa redução é de vital importância para o ambiente de computação móvel. No protocolo proposto, no caso de ocorrência de falhas a obtenção de estado global pode tornar-se mais cara devido a dependências transitivas. No entanto espera-se que o progresso da computação seja mais frequente do que a reconstruções de estados.

Esse artigo está estruturado da seguinte forma: na seção 2 apresentamos um modelo de sistema que é utilizado pelo protocolo proposto. Na seção 3 discutimos um problema inerente aos atuais mecanismos síncronos na detecção de estado global para o ambiente de computação móvel, e discutimos o protocolo desenvolvido por [AB94]. Na seção 4 apresentamos a técnica para manutenção de dependência desenvolvida por [JJ94]. Na seção 5 descrevemos o algoritmo que propomos para obtenção de estado global em ambiente de computação móvel. A seção 6 apresenta conclusões e perspectivas futuras de trabalho.

2 O modelo de sistema

O modelo de sistema utilizado pelo protocolo que propomos é baseado em [BBI94]. Nesse modelo, o termo móvel diz respeito à capacidade de mover-se enquanto mantém sua conexão com a rede, e um computador que possui esta capacidade é chamado de unidade móvel (mobile host - MH). Os sistemas que comunicam-se diretamente com os MHs são chamados de *mobile support stations* (MSS). Uma célula é uma área lógica ou geográfica sobre um MSS. Todos os MHs são associados a um MSS, e a ele são considerados locais. O MSS representa a localização corrente do MH na rede, e, portanto, um MH pode pertencer a somente uma célula em um dado instante do tempo. Além disso, um MH é sempre associado a um endereço fixo, seu *id*, independentemente da sua localização na rede.

O MH pode comunicar-se diretamente com um MSS (e vice e versa) somente se está fisicamente localizado dentro da célula do MSS. Para um MH h_i localizado em uma célula de um MSS m enviar uma mensagem, por exemplo, para um MH h_j que não está na mesma célula, ele deve contactar m para que o mesmo possa passar a mensagem para o MSS n local à célula de h_j. Em geral o MSS do MH origem não sabe a localização do MH destino, e deverá pesquisar na rede a sua localização.

Por motivo de simplificação, é assumido que todos os sistemas fixos são $MSSs$ e é usado o termo *sistema fixo* e MSS indistintamente. O modelo, portanto, consiste de dois conjuntos distintos de entidades: MHs e $MSSs$, com todos os sistemas fixos e o caminho de comunicação entre eles constituindo uma rede estática ou fixa, enquanto cada MSS e os MHs locais dentro de uma célula formam uma rede sem fio. A rede fixa garante entrega confiável e na ordem de envio entre quaisquer dois $MSSs$, com uma latência finita, mas aleatória. Na

rede sem fio, as mensagens são entregues na ordem FIFO entre os MHs. A Figura 1 abaixo ilustra o modelo do sistema.

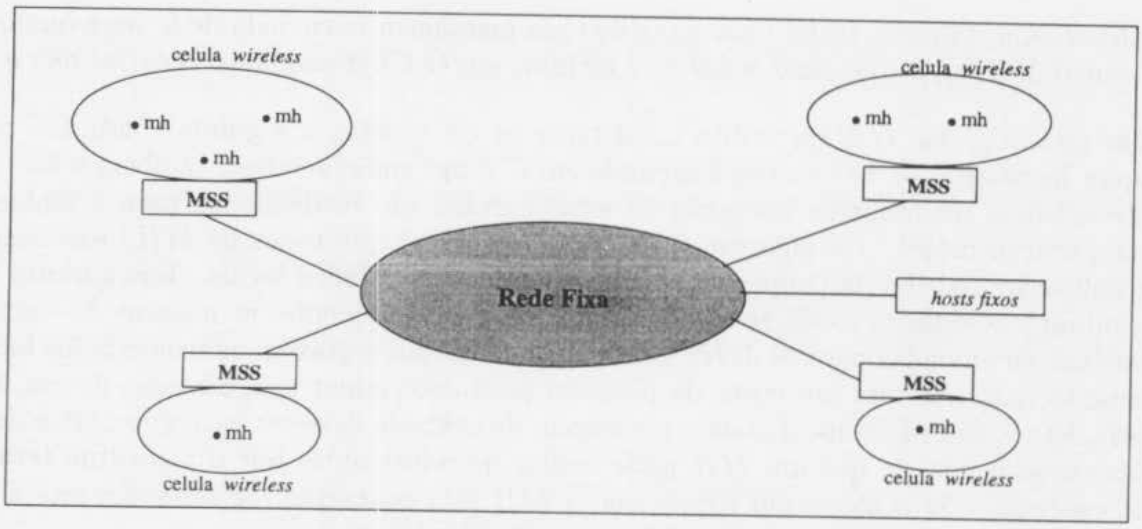

Figura 1: Modelo para algoritmos distribuídos

3 Motivação

Nessa seção descrevemos os problemas relacionados aos atuais mecanismos de obtenção de estado global que os tornam inadequados para o ambiente de computação móvel.

3.1 Os Algoritmos de detecção de estado global existentes

Algoritmos distribuídos para gravação de estado em sistemas distribuídos estáticos - *checkpoint* - têm sido largamente estudados, e as soluções existentes podem ser classificadas em duas categorias: síncronas e assíncronas. No esquema síncrono, todos os participantes cooperam na gravação de seus estados locais para que se possa garantir que o conjunto de estados locais seja consistente. Já no esquema assíncrono, cada participante pode escolher gravar seu estado local independente um do outro. Entretanto, um conjunto arbitrário de estados locais, um para cada participante, não forma necessariamente um estado global consistente. Assim sendo, esquemas de obtenção de estados globais assíncronos necessitam selecionar um conjunto de estados locais consistentes.

Para se modelar o estado global de uma aplicação distribuída executando nos sistemas móveis, assume-se que existe um canal FIFO C_{ij} de um MH h_i a um MH h_j. O estado global G_Ckpt da aplicação distribuída executando nos N MHs consiste de um estado local $local\_ckpt_i$ para cada MH h_i tal que:

- $local\_ckpt_i$ inclui um estado local do MH h_i. O estado local de um MH muda devido à recepção ou envio de uma mensagem, ou um evento local que não envolve comunicação.

Para um MH h_i com um estado inicial s_{i0}, o estado após uma seqüência de eventos E_i é representado por (s_{i0}, E_i); se h_i está gravando seu estado local, então todos os eventos e E_i são ditos incluídos no estado local e, portanto, no G_Ckpt.

- $local\_ckpt_i$ também inclui uma cópia de toda mensagem m enviada de h_i para qualquer outro MH h_j, se o evento $send(m)$ é incluído em G_Ckpt enquanto $recv(m)$ não o é.

Um estado global G_Ckpt é dito consistente se ele satisfaz a seguinte condição: para qualquer mensagem m, se $recv(m)$ é incluído em G_Ckpt então $send(m)$ também o é.

Mecanismos síncronos de obtenção de estado global são inadequados para o ambiente de computação móvel. Os algoritmos síncronos requerem que todos os MHs executando uma aplicação distribuída cooperem na gravação de seus estados locais. Isso garante que um conjunto de estados locais seja consistente. Consequentemente, mensagens de controle necessitam ser enviadas para os MHs a fim de que os mesmos gravem os seus estados locais. Entretanto, isso acarreta um custo de pesquisa para determinar a localização de um MH na rede, isto é, seu MSS local onde a mensagem de controle deve ser entregue. Além disso, acrescenta-se o fato de que um MH pode mudar de célula antes que o algoritmo termine a sua execução. Se o algoritmo requer que o MH seja contactado n vezes durante a sua execução, então o custo da pesquisa pode ser multiplicado por n.

Uma abordagem alternativa para pesquisar a localização corrente de um MH seria informar aos outros MHs interessados, toda vez que houver mudança na sua localização. Assim sendo, quando um MH necessitasse ser contactado durante a execução do algoritmo, sua localização corrente estaria disponível a todos os outros interessados. No entanto, se o MH muda com freqüência de célula e o número de MHs interessados na sua localização é grande, esta alternativa irá incorrer em um alto custo de informes. Portanto, algoritmos síncronos têm um alto custo de pesquisa/informe para o ambiente de computação móvel. Além disso, os algoritmos síncronos são seriamente afetados pela desconexão de um MH.

3.2 Um mecanismo para detecção de estado global para o ambiente móvel

No algoritmo proposto utilizaremos o esquema assíncrono para obtenção de estado global. Uma vez que os discos das unidades móveis não são considerados confiáveis, utilizaremos os discos dos $MSSs$ para armazenar os estados locais e os logs das mensagens dos MHs. No entanto, devido às mudanças de célula de um MH, sucessivos estados locais de um mesmo MH podem estar armazenados em diferentes $MSSs$. Além disso, a desconexão de um MH pode suspender a execução do algoritmo de obtenção de estados globais, até que o MH seja reconectado. Entretanto, a natureza voluntária de uma desconexão sugere que é possível um MH gravar seu estado local e transferi-lo para o seu MSS local como parte de um protocolo de desconexão. O algoritmo pode então usar esse estado como um substituto do estado local mais recente do MH na construção do estado global.

3.2.1 A Regra das duas fases

Um algoritmo de obtenção de estado global deve garantir que apesar de MHs gravarem seu estado local assincronamente, de acordo com algum protocolo local, é sempre possível

a obtenção de um corte consistente utilizando o conjunto de estados disponíveis. Dado um estado local qualquer de um MH, a partir do qual inicia-se o processo de obtenção de um estado global, o algoritmo deve ser capaz de selecionar um conjunto de estados locais, um para cada MH, de tal maneira que formem um estado global consistente. Para ilustrar o problema que surge quando cada MH grava seu estado local assincronamente, considere um sistema com três MHs h_1, h_2, h_3 mostrado na Figura 2 abaixo. Sejam C_1, C_2, C_3' os estados locais mais recentes dos MHs h_1, h_2, h_3 respectivamente. Seja C_3 o estado anterior do MH h_3 armazenado em um MSS diferente daquele de C_3'. Assuma agora que o algoritmo começa a construção do estado global a partir de C_3. A inclusão de C_3 no estado global implica que o evento $recv(M_2)$ seja incluído. Portanto, para manter um estado global consistente, o $send(M_2)$ deve ser incluído, ou seja, C_2. Uma vez que $recv(M_1)$ está incluído em C_2, e, portanto, no estado global, o evento $send(M_1)$ deve, também, ser incluído. O único estado em h_1 que inclui $send(M_1)$ é C_1, sendo esse selecionado como estado local representativo de h_1. Entretanto, a inclusão de C_1 implica que $recv(M_3)$ pertence ao estado global, e, portanto, o evento $send(M_3)$ deverá ser incluído para mantê-lo consistente. Entretanto, isso só é possível se o estado C_3' for incluído no estado global, representando o estado local de h_3 no lugar de C_3. Assim sendo, o algoritmo não é capaz de formar um estado global consistente a partir de C_3, ou seja, a partir de um estado local qualquer [AB94].

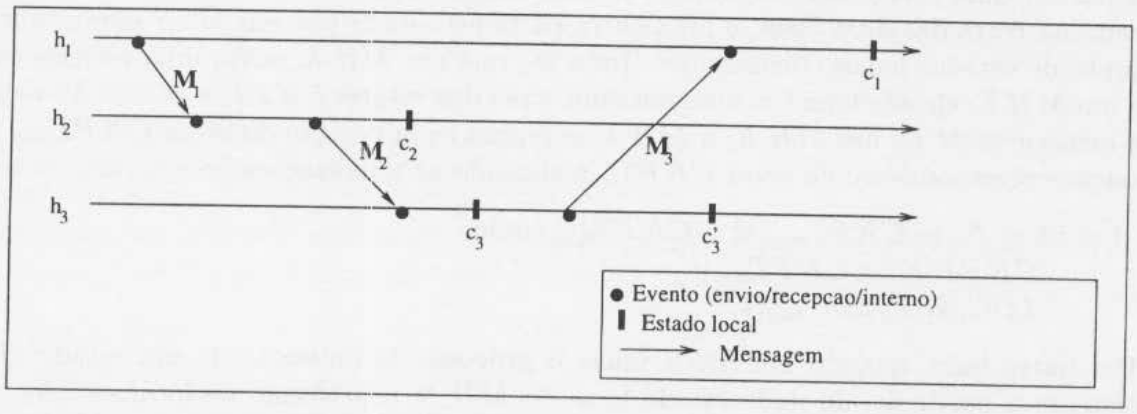

Figura 2: Gravação de estados locais assincronamente

A regra das duas fases proposta por [AB94] determina quando um MH h_i deve gravar seu estado local, e garante que nenhuma dependência é criada entre dois estados locais de um mesmo MH como no exemplo ilustrado acima. A regra funciona da seguinte forma:

- Se o MH h_i está na $fase_i == SEND$ e recebe uma mensagem M, então ele deve gravar seu estado local e mudar para a fase $RECV$, ou seja, $fase_i := RECV$ antes de entregar a mensagem para o aplicativo.
- Se o MH h_i está na $fase_i == RECV$, então, antes de enviar uma mensagem M para um outro MH, o MH h_i deve mudar para a fase $SEND$, isto é, $fase_i == SEND$.

O algoritmo proposto por [NF97] utiliza um mecanismo baseado na sincronização de relógios locais aos MHs. No entanto, esse mecanismo é muito simplista pois uma falha em

um desses relógios locais pode comprometer toda a execução do algoritmo.

3.2.2 O algoritmo de Acharya e Badrinath

O algoritmo desenvolvido por [AB94] não requer um controle explícito das mensagens enviadas para os MHs, evitando com isso o *overhead* do mecanismo síncrono. Ao invés disso, a cada mensagem é adicionada uma informação de controle, e cada MH grava seu estado local assincronamente.

Associado a cada $MH\ h_i$ existem dois vetores, $CKPT_i[1..N]$ e $LOC_i[1..N]$, ambos armazenados no MSS local ao $MH\ h_i$. $CKPT_i[i]$ é inicializado com 1, e as outras posições do vetor com zero. Sempre que h_i gravar seu estado local, ele incrementa o $CKPT_i[i]$. Portanto, $CKPT_i[i]$ sempre armazena o número do próximo estado local de h_i a ser gravado. $LOC_i[i]$ armazena o id do MSS local a h_i. Esse vetor muda sempre que h_i mover-se para uma nova célula. A regra que rege esses dois vetores é como se segue: Se $CKPT_i[j]$ é p e $LOC_i[j]$ é q, então um estado global que inclui $CKPT_i[i]$ como o representante do estado local de h_i, deve incluir o $p-esimo$ estado local do $MH\ h_j$ que está localizado no MSS cujo id é q. Além disso, cada MH obtém seu estado local sempre que mover-se para uma nova célula, antes de desconectar-se e quando requerido pela regra das duas fases. O algoritmo consiste de duas partes: uma executada por todo MH para gravar seu estado local independentemente baseado na regra das duas fases, e uma outra parte executada por um MSS para obter um conjunto de estados locais consistentes. Toda vez que um $MH\ h_i$ envia uma mensagem M para um $MH\ h_j$ ele adiciona à mensagem uma cópia dos vetores $CKPT_i$ e LOC_i. Ao receber uma mensagem M de um $MH\ h_j$ o $MH\ h_i$ compara cada posição do vetor $CKPT_{msg}$ com a posição correspondente do vetor $CKPT_i$, realizando as atualizações necessárias, ou seja:

$$1 \leq \forall k \leq N, \text{ se } CKPT_{msg}[k] > CKPT_i[j] \text{ então}$$
$$CKPT_i[k] := CKPT_{msg}[k]$$
$$LOC_i[k] := LOC_{msg}[k]$$

Por outro lado, quando um MSS inicia o processo de obtenção de um estado global consistente a partir de um dado estado local do $MH\ h_i$ consistindo de $local\_estado_i$, log_i, LOC_i e $CKPT_i$, ele envia uma mensagem $request(h_j, CKPT_i[j])$ para todos os MSS_q, onde q é o valor de $LOC_i[j]$, requisitando o estado local do $MH\ h_j$ cujo número é $CKPT_i[j]$.

4 Reduzindo o tamanho das mensagens

O algoritmo proposto utiliza a técnica de manutenção de dependência proposta por [JJ94] para reduzir o número de dependências transmitidas. Nessa técnica, desenvolvida para um sistema de depuração distribuída, os eventos podem ser adaptativamente observados enquanto é mantida a capacidade de recuperar todas as dependências casuais do evento observado. Jard-Jourdan definem uma relação pseudo-direta $<<$ nos eventos de uma computação distribuída, da seguinte maneira: se os eventos e_i e e_j acontecem nos processos p_i e p_j, respectivamente, então $e_j << e_i$ se existe um caminho de transferências de mensagem que inicia-se após e_j em p_j e antes de e_i em p_i, tal que não existe nenhum evento observado no caminho.

O relógio vetorial parcial p_vt_i no processo p_i é uma tupla da forma (j, v), indicando que o corrente estado de p_i é pseudo dependente de um evento, cujo número é v, no processo p_j. Inicialmente o relógio num processo p_i é: $p\_vt_i = \{(i, 0)\}$. Além disso, temos que:

1. Sejam $p\_vt_i = \{(i_1, v_1),...,(i, v),...\}$ denotando o corrente relógio vetorial parcial de p_i, e a variável e_vt_i sendo o timestamp do evento observado. Quando um evento é observado num processo p_i, as seguintes ações são executadas:
   - $e\_vt_i = \{(i_1, v_1),...,(i, v),...\}$.
   - $p\_vt_i = \{(i, v + 1)\}$.
2. Quando um processo p_j envia uma mensagem para p_i, ele adiciona o valor corrente de p_vt_j na mensagem.
3. Quando p_i recebe uma mensagem com um valor de relógio p_vt, p_i procede-se da seguinte maneira: Seja $p\_vt = \{(i_{m1}, v_{m1}), ..., (i_{mk}, v_{mk})\}$ e $p\_vt_i = \{(i_1, v_1), ..., (i_l, v_l)\}$. p_i atualiza o p_vt_i, onde p_vt_i é a união de:
   - todos (i_{mx}, v_{mx}) tal que $(i_{mx}, .)$ não aparece em p_vt_i,
   - todos (i_x, v_x) tal que $(i_x, .)$ não aparece em p_vt, e
   - todos $(i_x, max(v_{m1}, v_{xm}))$ para todo $(v_x, .)$ que aparece em p_vt e em p_vt_i.

5 O Algoritmo proposto

No algoritmo proposto, cada MH h_i mantém um relógio vetorial parcial h_vt_i. A tupla (j, v) desse relógio significa que o MH h_i é pseudo dependente de um estado local no MH h_j. h_vt_i é inicializado com $\{(i, 1)\}$, e sempre que um h_i gravar seu estado local, ele atualiza o seu relógio como definido pela técnica de Jard-Jourdan descrita acima. Assim sendo, $h\_vt_i[i]$ sempre armazena o próximo estado de h_i a ser gravado. Associado a cada MH h_i existe, também, um vetor loc_i que armazena as posições dos MHs h_j relacionados diretamente com o MH h_i através da relação pseudo-direta $<<$ descrita acima. $loc_i[i]$ armazena o id do MSS corrente a h_i. Esse vetor muda sempre que h_i move para uma nova célula. A regra que rege esses dois vetores é como se segue: Se $h\_vt_i[j]$ é p e $loc_i[j]$ é q, então um estado global que inclui $h\_vt_i[i]$ como o representante do estado local de h_i, deve incluir o p^{ht} estado local do MH h_j que está localizado no MSS cujo id é q. Além disso, cada MH obtém seu estado local sempre que mover para uma nova célula, antes de desconectar-se e quando requerido pela regra das duas fases. O algoritmo consiste de duas partes: uma executada por todo MH para gravar seu estado local independentemente baseado na regra das duas fases, e uma outra parte executada por um MSS para obter um conjunto de estados locais consistentes. Toda vez que um MH h_i envia uma mensagem M para um MH h_j ele adiciona à mensagem uma cópia dos vetores h_vt e loc . Note que a gravação de um estado local corresponde a um evento sendo observado. Por outro lado, quando um MSS inicia o processo de obtenção de um estado global consistente a partir de um dado estado local do MH h_i consistindo de $local\_estado_i$, log_i, loc_i e h_vt_i , ele envia uma mensagem $request(h_j, h\_vt_i[j])$ para todos os

MSS_q, onde q é o valor de $loc_i[j]$, requisitando pelo estado local do MH h_j cujo número é $h\_vt_i[j]$. Observe que a mensagem *request* é enviada apenas para os MSS que contém os MHs relacionados diretamente com o MH h_i.

5.1 Protocolo de gravação de estado

No envio de uma mensagem M

1. Se $fase_i$ é $RECV$, então $fase_i$ é mudada para $SEND$;
2. Adiciona uma cópia de h_vt_i e do loc_i à mensagem M;
3. Adiciona M ao log das mensagens, log_i.

No recebimento de uma mensagem M de um MH h_j

1. Se $fase_i == SEND$, então execute o procedimento *checkpointing*
2. $fase_i := RECV$
3. Seja $h\_vt = \{(i_{m1}, v_{m1}), \ldots (i_{mk}, v_{mk})\}$ e $h\_vt_i = \{(i_1, v_1), \ldots (i_l, v_l)\}$. Atualize o h_vt_i onde h_vt_i é a união de:
   - todos (i_{mx}, v_{mx}) tal que $(i_{mx},.)$ não aparece em h_vt_i,
   - todos (i_x, v_x) tal que $(i_x, .)$ não aparece em h_vt, e
   - todos $(i_x, max(v_{m1}, v_{xm}))$ para todo $(v_x, .)$ que aparece em h_vt e em h_vt_i.
4. Seja $L\_vt = \{(i_{m1}, mss_{m1}), \ldots, (i_{mk}, mss_{mk})\}$ e $loc_i = \{(i_1, mss_1), (i_l, mss_l)\}$. Atualize o loc_i, onde o loc_i é a união de:
   - todos (i_{mx}, mss_{mx}) tal que $(i_{mx}, .)$ não aparece em loc_i,
   - todos (i_x, mss_x) tal que $(i_x, .)$ não aparece em L_vt, e
   - todos (i_x, mss_{m1}) para todo $(v_x, .)$ que aparece em loc e em loc_i

Quando um evento é observado

1. Sejam $h\_vt_i = \{(i_1, v_1),\ldots,(i,v),\ldots\}$ denotando o corrente relógio vetorial parcial de h_i, a variável e_vh_i denotando o timestamp do evento observado e $loc_i = \{(i_1, mss_1),\ldots,(i,mss),\ldots\}$ denotando o vetor de posições correntes do MHs. Quando um evento é observado no MH h_i, as seguintes ações são executadas:
   - $e\_vh_i = \{(i_1, v_1),\ldots,(i,v),\ldots\}$,
   - $h\_vt_i = \{(i,v + 1)\}$,
   - $L\_vh_i = \{(i,mss)\}$.

5.2 Procedimento *checkpointing*

1. Gravar o estado local, $estado\_local_i$, da aplicação distribuída executando no h_i.
2. Transferir o estado local $estado\_local_i$, log_i, h_vt_i para o MSS local
3. $h\_vt_i[i] := h\_vt_i[i] + 1$

5.3 Protocolo de reconstrução de estado

Dado um estado local do MH h_i consistindo de $estado\_local_i$, log_i, e h_vt_i a partir do qual inicia-se o processo de obtenção de um estado global, um MSS executa os seguinte passos para obter um conjunto consistente de estados locais:

Iniciador

1. Para $1 \leq j \leq M$, requisite o estado local de h_j, cujo número é p, para o MSS_q através do envio de uma mensagem $request(h_j, h\_vt_i[j])$, onde M é o número de MHs h_j relacionados diretamente com o MH h_i, p é o valor de $h\_vt_i[j]$, e q é o valor de $loc_i[j]$.
2. O iniciador do processo espera até que receba um estado local de cada MH h_j e uma lista de dependência transitiva de cada um deles. Se houver algum MH h_k em alguma das listas recebidas para o qual ainda não foi enviada uma mensagem *request*, o iniciador repete o passo anterior. O iniciador espera até receber um estado local de todos os MSS para o qual ele enviou uma mensagem *request*. Se algum MSS responder que o estado local desejado não está disponível devido a uma falha do MH h_j, então o algoritmo de obtenção aborta. Se todos os MSS responderem com o estado local desejado, o conjunto desses estados locais formam um estado global consistente.

Respondendo a uma mensagem do tipo $request(h_j, h\_vt_i[j])$

1. Se o estado local do MH h_j com número p está disponível no MSS que recebeu a mensagem *request*(), então ele responde imediatamente ao iniciador do processo, enviando-lhe o estado requerido. Caso contrário, se o MH h_j ainda não gravou o próximo estado local que se iguale a p, o MSS espera até que o MH h_j obtenha o próximo estado local cujo número seja igual a p. O MH irá gravar e transferir o estado local requerido antes da sua próxima desconexão, ou antes de entregar ou receber qualquer mensagem. Entretanto, nenhum desses dois eventos são garantidos ocorrerem. Portanto, o MSS pode forçar o MH a gravar seu estado local. Além disso, é possível que h_j falhe antes de gravar o seu estado local. Nesse caso, o MSS informa ao inicializador a falha do MH h_j.

A Figura 3 abaixo ilustra o comportamento do algoritmo proposto. No melhor caso apenas um componente do relógio h_vt é enviado, no caso médio dois e no pior caso o relógio terá o mesmo tamanho que os vetores CKPT e LOC do algoritmo proposto por [AB94].

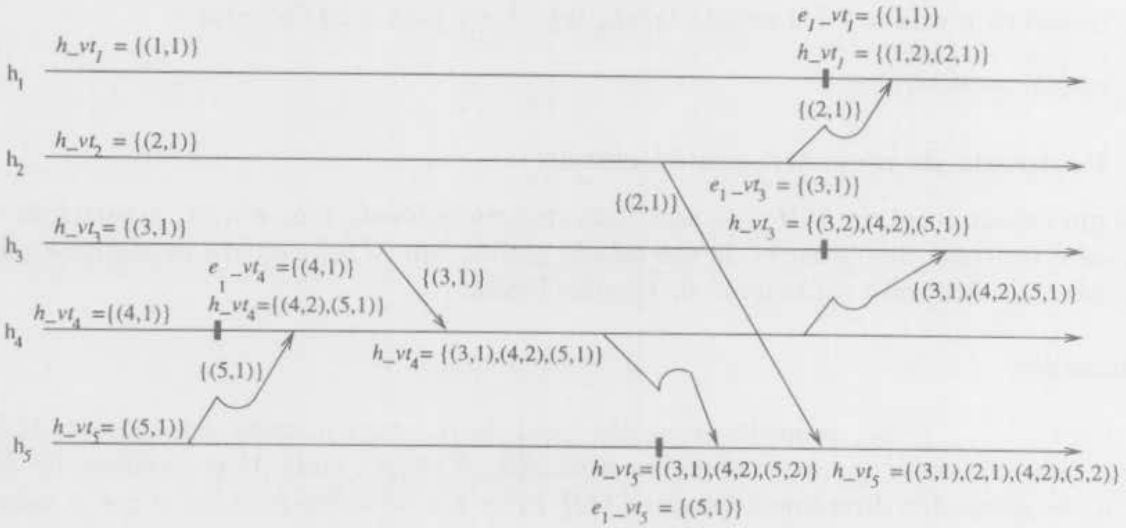

Figura 3: Exemplo de uma computação com o protocolo proposto

6 Conclusão

Nesse artigo apresentamos um novo protocolo de obtenção do estado global para o ambiente de computação móvel. Aplicando a técnica proposta por [JJ94], reduzimos o número de informações que são enviadas em uma mensagem em relação a outras abordagens. No protocolo proposto, um $MH\ h_i$ guarda somente as informações referentes aos MHs dos quais ele é pseudo-dependente num determinado estado corrente. Essas informações são armazenadas em um relógio vetorial parcial h_vt, e quando uma mensagem é enviada o $MH\ h_i$ adiciona à mensagem o valor corrente do relógio vetorial parcial h_vt. Com isso, temos no melhor caso o envio de somente um componente do relógio vetorial parcial, no caso médio dois componentes, e no pior caso o mesmo tamanho dos vetores CPKT e LOC do algoritmo desenvolvido por [AB94]. Levando-se em consideração as restrições de força, banda passante e latência nos sistemas ff móveis, essa redução é de vital importância para o ambiente da computação móvel. Deve-se notar que no algoritmo proposto para a obtenção do estado global, a reconstrução de estado pode tornar-se mais cara devido às dependências transitivas. Todavia, espera-se que o progresso da computação seja mais freqüente do que restaurações de estados.

Referências

[AB93] Arup Acharya and B.R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proc. of the third Intl. Conf. On Parallel and Distributed Information Systems*, May 1993.

[AB94] Arup Acharya and B.R. Badrinath. Checkpointing distributed aplications on mobile computers. In *Proc of the third Intl. Conf. On Parallel and Distributed*

Information Systems, Septembr 1994.

[BBI94] Arup Acharya B.R. Badrinath and T. Imielinski. Designing distributed algorithms for mobile computing networks. In *Proc. of the fourth Intl. Conf. On Distributed Conputing Systems*, May 1994.

[FZ94] George H. Forman and John Zahorjan. The challenges of mobile computing. In *Available as UW CSE Tech Report 93-11-03 from ftp.cs.washington.edu*, March 1994.

[HKT93] Lars Krombholz Henning Koch and Oliver Theel. A brief introduction into world of mobile computing. In *Department of Computer Science, University of Darmstad*, Alexanderstr 10, D-W-6100 Darmstadt, Germany, May 1993.

[HL95] P. Honeyman and L.B.Huston. Communications and consistency in mobile file systems. In *CITI Technical Report in IEEE Personal Communications, Special Issue on Mobile Computing*, December 1995.

[IB93] Tomasz Imielinski and B.R. Bandrinath. Data management for mobile computing. In *SIGMOD RECORD*, volume 22, No 1, pages 34–39, March 1993.

[JJ94] C. Jard and G-C. Jourdan. Dependency tracking and filtering in distributed computations. In *A full presentation appeared as IRISA Tech Report No. 851*, 1994.

[KS91] J.J. Kistler and M. Satynarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symp. on Operating System Principles*, pages 213–225, October 1991.

[Kue97] Geoffrey Houston Kuenning. Seer: predictive file hoarding for disconnected mobile operation. *University of California, Los Angeles*, 1997.

[Nar93] Vivek R. Narasayya. Distributed transactions in a mobile computing system. In *Subnitted as part of requirement for CSE552*, October 1993.

[NF97] Nuno Neves and W. Kent Fuchs. Adaptive recovery for mobile environments. In *Communications Of The ACM*, volume 40, pages 68–74, January 1997.

[RRPK97] David Ratner, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kenning. Replication requirements in mobile environments. *1st international workshop on discrete algorithms and methods for mobile computing and communications*, October 1997.