ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

به نام خداوند مهربان

تمرین دوم سیستم عامل 2 محمد تنهایی 87202446 نرم افزار

1. در الگوریتم Lamport-Shostak-Pease OM((n-1)/3) اجرا می شود ، حداکثر تاخیر برای رسیدن به توافق برابر است با بیشترین هزینه اجرا در یک پردازه درست (که برابر 2 است) + بیشترین هزینه ارسال که برابر 1 واحد است ، هر بار om(m) در مرحله k، به تعداد n-k تا پیغام om(m-1) به بقیه می فرستد و هزینه ارسال برابر با n-k و هزینه اجرا نیز برابر است با 2 پس om(m) = 2+(n-k)om(m-1) اگر از مقدار 2 صرف نظر کنیم برای هر پردازنده داریم :

$$O(m) = \frac{(n-1)(n-2)(n-3)\dots(n-m-1)+(n-1)(n-2)\dots(n-m)+\dots+(n-1)}{n}$$

و این پیچیدگی زمانی بسیار بالایی است.

اما حداقل تاخیر رسیدن به توافق برابر است با کمترین هزینه ارسال که برابر صفر است + کمترین هزینه اجرا در پردازه درست ، که برابر می شود با om(m) = 1+om(m-1) و در نتیجه om(m) = m = (n-1)/3

پیچیدگی پیغامی نیز برابر است با (n-1)(n-2)(n-3)…(n-m-1)+(n-1)(n-2)…(n-m)+⋯+(n-1)

که برابر می شود با $O(n^m)$

در الگوریتم Dolev الگوریتم به 2m+3 دور برای رسیدن به توافق احتیاج دارد ؛ در هر دور بیشترین هزینه اجرا برابر است با بیشترین هزینه یک پردازنده که برابر است با 2 + بیشترین هزینه ارسال که برابر 1 است و در نتیجه ماکزیمم هزینه این الگوریتم برابر است با2n+7 = 2(n-1)/3+9 *(2+1) = (2+1)*2m+9

حداقل تاخیر رسیدن به توافق در این حالت نیز برابر است با کمترین هزینه ارسال + کمترین هزینه اجرا در پردازه درست که برابر می شود با 2(n-1)/3+3 = 2*m+9

پیچیدگی پیغامی الگوریتم در بدترین حالت به صورت زیر است : هر پردازنده از همه بقیه پردازنده ها پیغام مبنی بر شاهد بودن بقیه پردازنده ها دریافت کند ، در این حالت هر پردازنده می تواند تعداد (n+1)*(n-1) تا پیام از بقیه دریافت کند n+1 به خاطر اینکه می تواند پیام * یا k را دریافت کند و n-1 به خاطر اینکه می تواند از n-1 پردازنده این پیغام را دریافت کند ، در کل تعداد $O(n^3)$ پیغام می تواند مبادله شود که بسیار بسیار کمتر از الگوریتم قبلی است!

---

2. توافق Byzantine یک نمونه خاص از interactive consistency است که در آن ، تنها به مقدار یک پردازنده توجه می شود ، به عبارت بهتر اگر فرض کنیم که فقط یک پردازنده مقدار ارائه کند به Byzantine می رسیم .

اگر هر یک از n پردازنده یک کپی از توافق Byzantine را اجرا کنند به توافق interactive consistency می رسیم.

مشکل consensus می تواند با جواب interactive consistency حل شود .

راه حل های interactive consistency و consensus می تواند از طریق توافق Byzantine به دست آید ، به عبارت بهتر Byzatine agreement یک primitive برای دو مسئله بالاست.

بنابراین هر سه مسئله مشابه و مثل هم هستند و می توان با داشتن یکی بقیه را نیز شبیه سازی کرد.

.3

The transmitter (node p0) signs its value and sends to all other nodes.

For each i: مرحله اول

 If node pi receives a message of the form (a:0) from the transmitter then

- It sets Vi to {a}.
- It sends the message (a:0,i) to all other nodes.

 If node pi receives a message of the form (a:0,j1,j2,..,jk) and a is not in Vi,

Then: مرحله دوم

- It adds a to Vi
- If k<m, it sends the message (a:0,j1,j2,…,jk,i) to every node other than j1,j2,…,jk

For each i, مرحله سوم

 When node pi receive no more messages, it considers the final value as choice (Vi).

به نظر می رسد که مقدار m در خط 5 تا به آخر صحیح تر از n است

الف) با 2 فرض می توانیم این کار را انجام دهیم :

فرض کنیم که P_0 بی خطا است ، در نتیجه بعد از ارسال پیام v:0 در مرحله اول ، در مرحله دوم همه پردازنده ها پیام v را که دریافت کرده اند برای هم می فرستند .دو پردازنده خطا دار به صورت v':i پیغام خود را ارسال می کند ، همه پردازنده های سالم این پیغام را دریافت می کنند ، چون که قبلاً دستور v را از P_0 دریافت کرده اند ، آنرا نادیده می گیرند! ، و همه پردازنده های درست روی v به توافق می رسند .

فرض کنیم که P_0 خطا دار است ، در نتیجه یک مقدار v:0 را برای عده ای از پردازنده ها و مقدار v':0 را برای عده ای دیگر می فرستد ، پردازنده های درست در مرحله دوم مقدار دریافتی را برای نود های دیگر ارسال می کنند (به جز خودشان و P_0) ، تنها پردازه ناسالم هم پیغام V':i را جعل می کند ، در مرحله بعد (مرحله سوم) همه پردازنده های سالم به خطا دار بودن P_0 پی می برند ، زیرا هر پردازنده همه پیغام هایی که از P_0 صادر شده را دریافت می کنند ، مجموعه V نیز در همه پردازنده یکسان است ، بنابراین در مرحله سوم پردازنده ها به توافق می رسند.

به زبان بهتر :[هر کدام از پردازنده های سالم i و j در مرحله سوم از یک فرمان اطاعت می کنند. اگر i مقدار پیشنهادی v را در مرحله دوم دریافت کند ، آنگاه آن را در مرحله دوم برای j ارسال می کند . در نتیجه j در مرحله دوم این مقدار را دریافت می کند ، چون که i به مجموعه دریافتی های j اضافه شده است در نتیجه j پیغام را برای بقیه نودهایی که در لیست دریافتی آن نیستند می فرستد ، پس هر پردازنده سالمی اگر مقداری را به v اضافه کند ، پردازنده های سالم دیگر نیز اضافه خواهند کرد . و می توانند روی مقدار موجود توافق کنند] اثبات درستی کل الگوریتم :

ثابت می کنیم که برای هر m الگوریتم SM(m) مسئله byzantine را با m پردازنده خطا دار حل می کند:

با استدلالی مشابه بالا اگر پردازنده صفر سالم باشد ، در مرحله اول پیغام امضا شده خود را به صورت V:0 به همه همسایگان می فرستد ، هر همسایه سالم پیغام را در مرحله دوم دریافت می کند ، اگر پردازنده همسایه خطاداری وجود نداشته باشد که پیغام V':i را جعل کند ، پردازنده سالم پیغام اضافی ای در مرحله دوم دریافت نمی کند ، بنابراین برای هر پردازنده سالم i ، V_i حفظ می شود . و V_i تنها شامل یک عضو است، همه روی مقداری که پردازنده سالم اولیه فرستاده به توافق می رسند .

فرض می کنیم که پردازنده صفر خطادار باشد ، هر کدام از پردازنده های سالم i و j در مرحله سوم از یک فرمان P_0 اطاعت می کنند.اگر هرکدام از آنها در مرحله دوم یک مجموعه V_j و V_i را دریافت کرده باشند .در نتیجه برای اثبات درستی باید ثابت کنیم که اگر i مقدار v را در مرحله دوم به V_i اضافه کند ،j نیز مقدار v را در مرحله دوم به V_j اضافه خواهد کرد. اگر i مقدار v را در مرحله دوم دریافت کند ، آنگاه آن را در مرحله دوم برای j ارسال می کند . در نتیجه j در مرحله سوم این مقدار را دریافت می کند ، چون که v به مجموعه دریافتی های j اضافه شده است در نتیجه j پیغام را برای بقیه نودهایی که در لیست دریافتی آن نیستند می فرستد ، اگر i مقدار v را به به ترتیب موجود اضافه کرده باشد در نتیجه باید پیغامی به شکل $v:0:j_1,j_2,\ldots,j_k$ دریافت کند ، اگر j یکی از j_r ها باشد که قبلاً پیغام i را دریافت کرده است ، اما اگر j_k جز این j_r ها نباشد 2 حالت داریم :

k<m که در نتیجه i پیغام $v:0:j_1,j_2,\ldots,j_k$ را برای j_k ارسال می کند و j_k آنرا دریافت می کند.

k=m این حالت نشان می دهد که P_0 ناسالم است ، در نتیجه حداکثر m-1 تا از همسایگان ناسالم هستند ، حداقل یک پردازنده سالم وجود دارد ، این پردازنده سالم حتماً قبلاً مقدار v که دریافت کرده است را برای j فرستاده است ، پس j حتماً قبل از این مرحله مقدار v را از یک پردازه درست دریافت کرده است.■

ب) برای هر دنباله ای از $j_1,j_2,\ldots,j_k$ که $k \le m$ در مرحله دوم حداکثر یک پیغام به شکل $v:0:j_1,j_2,\ldots,j_k$ دریافت می شود ، برای k'<k برای هر دنباله از $j_1,j_2,\ldots,j_{k'}$ نیز حداکثر یک پیغام به شکل $v:0:j_1,j_2,\ldots,j_{k'}$ دریافت می شود (یک دور) ، اگر k را هر دفعه کم کنیم می بینیم که برای مجموعه سایت هایی با m پردازنده خطا دار ، حداکثر m تا دور برای دریافت همه پیغامها طول می کشد ، که اگر پیغام v:0 را که در ابتدای الگوریتم ارسال می شود را نیز محاسبه کنیم می بینم که حداکثر تعداد دورها برابر با m+1 است.

ج) فرض کنیم که شرط برقرار نباشد ، در نتیجه یا n=m است ، یا n=m+1 است ، به عبارت بهتر یا هیچ پردازنده سالمی نداریم ، یا یک پردازنده سالم داریم ، اگر هیچ پردازنده سالمی نداشته باشیم که توافق مطرح نیست ، اگر یک پردازنده سالم داشته باشیم که بر روی مقدار خودش تصمیم می گیرد ، و باز هم توافق مطرح نیست ، حالت دیگری غیر از این حالت نداریم ، بنابراین برای رسیدن به توافق حتماً باید n > m+1 باشد.

---

4. **الف)** فرض کنیم که A و B پردازه های درستی باشند ، A پیغام همه پخشی را دریافت کرده و به همه می فرستد ، B این پیغام را می گیرد و به همه کسانی که همسایه آن هستند می فرستد ، اما C که یک همسایه آن است به دلیل خرابی کانال پیغام غلط m' را دریافت می کند ، در نتیجه همه پردازنده های درست پیام m را دریافت کرده اند پس

R.Bcast داریم ، اما چون که C ا پیام m' را دریافت کرده است و پردازنده های درست آنرا دریافت نکرده اند ، Uniform نیست.

ب) A یک پیغام می فرستد، B روی سایت دیگری پیغام A را گرفته و جواب آنرا ارسال می‌کند، C پیغام B را قبل از پیغام A دریافت می‌کند. در این حالت ترتیب FIFO رعایت شده است ؛ اما ترتیب causal رعایت نشده است.

---

5. بله می شود ، اگر یک پردازنده خطا دار مانند P_i به پردازنده سالمی مانند P_j پیغام m را بدهد و P_j این پیغام را deliver کند و به هیچ کدام از پردازنده های سالم نفرستد ، همه شروط مطرح شده را داریم اما شرط توافق را نداریم . چنین الگوریتمی را به سادگی می توان طراحی کرد.

---

6. در الگوریتمی که در درس مطرح شده ، هر پردازنده همه پیغام هایی که تاکنون (سابقه پیغامی) دریافت کرده است را برای همه پردازنده ها ارسال می کند ، ارسال همه پیغام های دریافت شده به همراه پیغام جدید سربار زیادی برای سیستم دارد . پیچیدگی پیامی الگوریتم causal Bcast مطرح شده با افزایش تعداد پیام های تحویل داده شده افزایش می یابد . به عنوان نمونه پیچیدگی پیغامی در زمان تحویل پیغام m ام برابر است با $CB(m) = n * (m-1)$ به عبارت بهتر پیچیدگی کل الگوریتم برای تحویل m پیغام برابر است با $n*(1+2+3+4+\ldots+m)$ که برابر است با $n*(m+1)*m/2$ که در کل برابر می شود با $O(nm^2)$

....

به جای این روش از روش دیگری به نام Vector Clock استفاده می شود . در Causal delivery ما چنین الگوریتمی را طراحی کردیم مجدداً از روش مشابه برای طراحی الگوریتم خود استفاده می کنیم ،

فرض کنیم که ReliableBroadcast (rb) را داریم :

rcoBroadcast به معنای RelabelCausalOrderBroadcast

init : for all pi in S: $VC[P_i] := 0$;

upon event < RelabelCausalOrderBroadcast, m> **do**

VC[self] := VC[self] + 1.

trigger < RelabelBroadcast, [Data,VC,m]>;

upon event < ReliableBroadcastDeliver, P_j, [Data,VCm,m]> **do**

wait until ($(VC[P_j] \geq VCm[P_j] - 1)$ and

(**for all** $P_k \neq P_j$: $VC[p_k] \geq VCm[P_k]$));

trigger < RelabelCausalOrderDeliver, p_j,m>;

if $P_j \neq$ self **then**

$VC[P_j] := VC[P_j] + 1$.

همان طور که می بینید برای پیاده سازی از یک حلقه wait نیز استفاده شده است ، این حلقه تا زمانی توقف می کند که نوبت تحویل پیغام P_j شود .

---

7. این مورد برای الگوریتم لمپورت بدیهی به نظر می رسد ، هر سایت یک $request_queue_i$ دارد که درخواست‌ها به ترتیب زمان مهر در آن قرار دارند ، در شرایط زیر یک پردازنده وارد CS می شود :
   1. S_i درخواست (پیغام) با زمان مهر بزرگتر از (ts_i,i) از همه دیگر سایت‌ها دریافت کرده باشد.
   2. درخواست S_i در سَرِ صفِ $request_queue_i$ باشد.

در حالت اول چون که زمان پردازنده از همه کوچکتر است وارد CS می شود و در حالت دوم چون که $request_queue_i$ به ترتیب زمان مهر ها مرتب است بنابراین زمان مهر پردازنده سر صف از همه کوچکتر است ، نتیجه منطقی از این 2 حالت این است که همیشه پردازنده با زمان مهر کمتر قبل از پردازنده با زمان مهر بیشتر وارد CS می شود و به زبان ساده تر: پردازنده ها به ترتیب غیر نزولی (صعودی) زمان مهر وارد CS می شوند.

در الگوریتم Ricart-Agrawala هر پردازنده در صورتی وارد CS می شود که از همه سایت های R_i ، reply دریافت کرده باشد . reply در صورتی ارسال می شود که زمان مهر درخواست کننده کمتر از زمان مهر سایت Ri باشد ، بنابراین هر بار تنها یک سایت که زمان مهر آن از همه کمتر است از همه Ri ها reply دریافت می کند و این به معنای این است که پردازنده ها بر اساس زمان مهر صعودی وارد CS می شوند.

در مورد الگوریتم میکاوا چنین قضیه ای صادق نیست ، فرض کنید که 3 پردازنده داریم ، در زمان t پردازنده 2 درخواست خود را برای پردازنده 1 که در R_i آن است می فرستد . در زمان t+1 پردازنده 1 درخواست خود را به پردازنده سوم که در حال اجرای cs است می دهد ، پس از بازگشت از CS پردازنده 3 پیغام reply را به پردازنده 1 می فرستد و پردازنده 1 وارد CS می شود در حالی که زمان مهر درخواست پردازنده 2 کمتر از زمان مهر درخواست پردازنده 1 بوده است ، پس از پردازنده 1 ، پردازنده 2 وارد CS خواهد شد . بنابراین الگوریتم میکاوا ترتیب زمان مهر درخواست ها را رعایت نمی کند.

R1 = {1, 3}; R3 = {2, 3}; R2 = {1, 2}

---

8. **الف)** در حالتی که بار سیستم کم باشد ، فرض می کنیم که تعداد درخواست های ورود به CS برابر تعداد k باشد ، در این حالت : زمان پاسخ برابر است با N/2+E*k ، تاخیر همگامی برابر با (N/k)*T است و کارایی برابر است با فرستادن

N پیغام در بدترین حالت و N/k پیغام در بار کم. همان طور که می بینیم در بار کم : زمان پاسخ کم است تقریباً N/2+E ، تاخیر همگامی زیاد است و فاصله بین دو ورود به CS تقریباً برابر با NT است ، و کارایی نیز در این حالت پایین است : N پیغام برای یک بار ورود.

ب) در حالتی که بار سیستم زیاد باشد : فرمول های قبل برای K جدید نیز صادق هستند ، در این حالت زمان پاسخ زیاد است N/2+NE ، تاخیر همگامی برابر T است و کارایی نیز در این حالت بالا است 1 پیغام برای هر بار ورود لازم است. البته به شرطی که K به N نزدیک باشد.

همه حرف هایی که زدیم در صورتی درست است که مهره گم نشود!

---

9. **الف)** برای الگوریتم سوزوکی و کاسامی مقادیر LN و RN_i طی درخواست ها به صورت زیر است :

1.مهره در اختیار سایت s1 است ،فرض می کنیم که از CS خارج شده است:

$$LN = [1 \quad 0 \quad 0 \quad 0]$$

$$RN = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

2.سایت S2 درخواست ورود به CS را به همه می فرستد ، ابتدا شماره ترتیبی خود را یکی زیاد می کند ، سپس درخواست را برای همه سایت های دیگر می فرستد ، سایت های دیگر $Rn_i[j]$ را افزایش می دهند . مهره در اختیار s1 است. بعد از آزاد شدن مهره ، سایت S1 مهره را در اختیار سایت S2 قرار می دهد زیرا $RN_1[j]=LN[j]+1$

$$LN = [1 \quad 0 \quad 0 \quad 0]$$

$$RN = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

3.سایت S4 درخواست ورود را بعد از خروج S2 می دهد ، باز شماره ترتیبی خود را افزایش می دهد ، چون که S2 خارج شده است و مهره آزاد است آنرا برای S2 می فرستد .

$$LN = [1 \quad 1 \quad 0 \quad 0]$$

$$RN = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

4.سایت S1 در خواست ورود به CS می دهد ، اما هنوز S4 از ناحیه بحرانی خارج نشده است ، در این حالت باز شماره ترتیبی خود را زیاد می کند و درخواست را به همه سایت ها می دهد ، S1 در صف مهره قرار می گیرد.

$$LN = [1 \quad 1 \quad 0 \quad 0]$$

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

$$RN = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \end{bmatrix}$$

5. در نهایت سایت S4 از ناحیه CS خارج می شود و مهره در اختیار سایت S1 قرار می گیرد:

$$LN = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}$$

$$RN = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \end{bmatrix}$$

ب) برای الگوریتم سینگهال ساختمان های داده در ابتدا به صورت زیر هستند :

$$TSV = \begin{bmatrix} H & N & N & N \end{bmatrix} \qquad TSN = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$SV = \begin{bmatrix} H & N & N & N \\ R & N & N & N \\ R & R & N & N \\ R & R & R & N \end{bmatrix} \qquad SN = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

1.سایت S2 درخواست ورود به CS دارد ، ابتدا شماره ترتیبی خود را یکی زیاد می کند و حالت خود را به R تغییر مـی دهد، سپس درخواست را برای سایت S1 می فرستد،

$$TSV = \begin{bmatrix} H & N & N & N \end{bmatrix} \qquad TSN = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$SV = \begin{bmatrix} H & N & N & N \\ N & R & N & N \\ R & R & N & N \\ R & R & R & N \end{bmatrix} \qquad SN = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

2.سایت S2 توکن را از سایت S1 دریافت کرده و به حالت E می رود.

$$TSV = \begin{bmatrix} N & H & N & N \end{bmatrix} \qquad TSN = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$$

$$SV = \begin{bmatrix} N & R & N & N \\ N & E & N & N \\ R & R & N & N \\ R & R & R & N \end{bmatrix} \qquad SN = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

3.سایت S4 درخواست ورود را بعد از خروج S2 می دهد ، باز شماره ترتیبی خود را افزایش می دهد ، حالت خـود را بـه R تغییر داده و برای سایت های S1و S2 و S3 درخواست خود را می فرستد.

$$TSV = \begin{bmatrix} N & H & N & N \end{bmatrix} \qquad TSN = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$$

$$SV = \begin{bmatrix} N & R & N & N \\ N & H & N & N \\ R & R & N & N \\ N & N & N & R \end{bmatrix} \qquad SN = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

4.حال سایت S2 که مهره را در اختیار دارد ، آنرا برای S4 می فرستد، بقیه سایت ها هم حالـت S4 را در وکتـور حالـت خود به R تغییر می دهند:

$$TSV = [N \quad N \quad N \quad H] \qquad TSN = [1 \quad 1 \quad 0 \quad 1]$$

$$SV = \begin{bmatrix} N & R & N & R \\ N & N & N & R \\ R & R & N & R \\ N & N & N & E \end{bmatrix} \qquad SN = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

5. سایت S1 در خواست ورود به CS دارد ، S4 هنوز در حالت E قرار دارد ، S1 شماره ترتیبی خود را افـزایش داده و درخواست خود را برای سایت های S2 و S4 می فرستد :

$$TSV = [N \quad N \quad N \quad H] \qquad TSN = [1 \quad 1 \quad 0 \quad 1]$$

$$SV = \begin{bmatrix} R & N & N & N \\ N & N & N & R \\ R & R & N & R \\ N & N & N & E \end{bmatrix} \qquad SN = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

6.سایت S4 در حال اجرا است ، بنابراین وضعیت S1 را برابـر R قـرار مـی دهـد ، سـایت S2 نیـز در حالـت N اسـت بنابراین این سایت نیز وضعیت S2 را در رکورد خود برابر R قرار می دهد ، سایت S1 به صف مهره اضافه می شود:

$$TSV = [N \quad N \quad N \quad H] \qquad TSN = [1 \quad 1 \quad 0 \quad 1]$$

$$SV = \begin{bmatrix} R & N & N & N \\ R & N & N & R \\ R & R & N & R \\ R & N & N & E \end{bmatrix} \qquad SN = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \end{bmatrix}$$

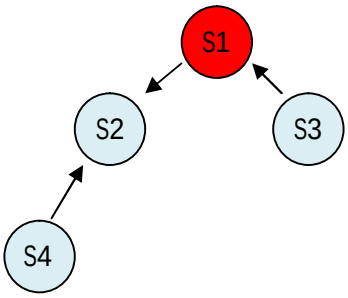
7.سایت S4 از CS خارج می شود ، SN خود را به روز می کند ، و مهره را برای عنصر سر صف یعنی S1 می فرستد:

$$TSV = [H \quad N \quad N \quad N] \qquad TSN = [2 \quad 1 \quad 0 \quad 1]$$

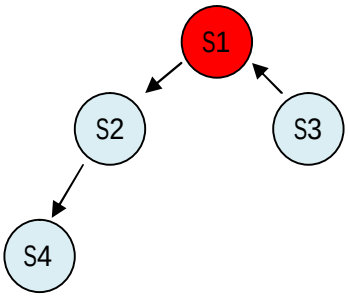
$$SV = \begin{bmatrix} H & N & N & N \\ R & N & N & R \\ R & R & N & R \\ R & N & N & N \end{bmatrix} \qquad SN = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 \end{bmatrix}$$

ج) با توجه به درخت و درخواست ها روند اجرای الگوریتم به صورت زیر است :

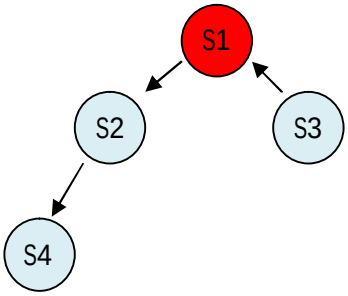
1.سایت S2 درخواست خود را برای S1 می فرستد ، و این درخواست را در request-q_2 قرار می دهد ، S1 با دریافـت این درخواست ، holder را به S2 تغییر می دهد و مهره را برای S2 ارسال می کند ، S2 چون که درخواسـت خـود را سر صف می بیند ، این درخواست را حذف کرده و به CS می رود.S2 پس از خروج مهره را نزد خود نگه می دارد ، زیـرا صف درخواست های آن خالی است.



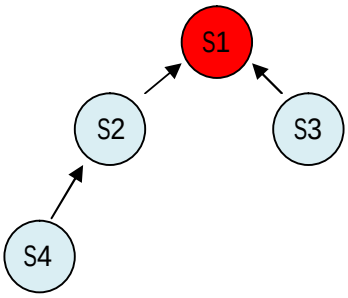
S4.2 درخواست ورود به CS دارد ، بنابراین درخواست خود را در $request\text{-}q_4$ قرار می دهد و درخواستی را برای S2 می فرستد ، چون که S2 مهره را در اختیار دارد و هیچ requestای در صف آن قرار ندارد تنهای holder را به S4 مقدار دهی می کند و مهره را به S4 می فرستد ، S4 با دریافت مهره ، درخواست خود را سر صف می بیند ، به حالت CS می رود.



3.حال S1 درخواست ورود به CS دارد ، برای این منظور درخواست خود را در صف $request\text{-}q_1$ قرار می دهد و درخواست مهره را برای S2 ارسال می کند ، S2 درخواست S1 را در صف خود قرار داده و یک درخواست برای S4 ارسال می کند ، S4 در حال اجرای CS است بنابراین هنوز هیچ جوابی به این درخواست نمی دهد.(اما درخواست را در صف درخواست های خود قرار می دهد)



S4.4 پس از خروج از CS به سر صف خود نگاه می کند ، مهره را برای S2 که متقاضی مهره بوده است می فرستد و holder خود را به S2 تنظیم می کند، S2 با دریافت مهره ، به سر صف خود نگاه کرده و آنرا برای S1 می فرستد ، S2 نیز holder خود را به S1 تنظیم می کند ، چون که خود S2 و S4 درخواست مهره ای نداشته اند ریال صف آنها پس از ارسال مهره مجدداً خالی می شود ، حال مهره در اختیار S1 است و می تواند وارد CS شود.



10. در الگوریتم Ricart-Agrawala پس از هر بار خروج از CS یک reply به همه سایت های دیگر ارسال می شود ، ممکن است که هیچ سایت دیگری درخواست نداشته باشد ، reply به همه ارسال می شود ، سایتی که CS را در اختیار داشته برای ورود مجدد به CS باید از همه سایت های دیگر reply دریافت کند ،علاوه بر این که یک سربار پیامی اضافی به اندازه 2N به سیستم تحمیل می شود ، تاخیر پیامی به اندازه 2T را نیز باید تحمل کنیم اما راه حل چیست؟
یک راه حل این است که ارسال reply را تا دریافت یک request جدید به تاخیر بیاندازیم ، هنگامی که یک request جدید از سایت های دیگر دریافت کردیم و از CS خارج شده بودیم یک reply برای همه سایت های دیگر می فرستیم .
در الگوریتم جدید اگر درخواست جدیدی از سایر سایت ها دریافت نکنیم ، الگوریتم می تواند چندین بار وارد CS شود ، الگوریتم خاصیت Liveness را دارد ، فرض کنیم که سایت جدید S' تقاضای ورود داشته باشد ، اگر تقاضای ورود آن به دست سایت S برسد و زمان مهر آن کمتر از درخواستهای دیگر سایت ها باشد ، سایت S پس از خروج از CS ، reply را برای همه سایت های دیگر می فرستد . بنابراین در صورت وجود سایت متقاضی حتماً وارد CS می شود (دیر یا زود!!)

والسلام