Uppsats i kursen Parallell Simulering
Institutionen för Telekommunikations- och Datorsystem
Kungliga Tekniska Högskolan

Stockholm 910531

Matematikerlinjens
Datalogiinriktning

Patrik Fältström
Patron Pehrs Väg 27
141 35 HUDDINGE

Telefon: 08-608 08 33
Internet: paf@nada.kth.se

Innehåll

1. Abstract

Parallell händelsestyrd simulering har varit av mycket stort intresse bland forskarna under de senaste åren. Detta dels för att massparallella datorer har blivit billigare och dels för att det har visat sig att simuleringstiden kan minskas drastiskt på detta sätt.

En vanlig metod att hantera parallell händelsestyrd simulering är enligt den Konservativa CM-metoden som framlades av Chandy och Misra i november 1981. Denna beskrivs i denna uppsats tredje del. De första två behandlar händelsestyrd simulering och parallell händelsestyrd simulering för att ge en bakgrund till resonemanget i tredje delen.

Uppsatsen är ett delmoment i kursen "Parallell simulering" som hölls på Institutionen för Telekommunikation och Datorsystem våren 1991.

Stockholm 910531
Patrik Fältström

2. Händelsestyrd simulering

Simulering är definierat som utvecklingen av en matematisk-logisk modell av ett system och en experimentell förändring av indata till modellen på en dator. Modellen är en förenklad bild av det system man vill studera. Vid en simulering skapar man sig en bild av hur systemet fungerar eller kommer fungera. Därför är även skapandet av modellen en viktig del av simuleringen. Det är detta som bestämmer hur relevant resultatet blir på det simulerade systemet.

Simuleringen kan delas in i flera olika steg:

1) **Problemformulering** - som innebär att man definierar de delar av det ursprungliga systemet som man är intresserad av.
2) **Målinriktning** - en skiss av de frågor som man vill ha besvarade under simuleringen. Detta innebär en ytterligare definiering av vilka delar av systemet man ska inrikta sig på.
3) **Byggande av modellen** - den slutgiltiga processen när man definierar de olika delarna av modellen, dess karaktäristika, samt hur de olika delarna ska interagera.
4) **Datainsamling** - insamling av data i systemet som ger modelleraren en bild av hur de olika delarna i modellen uppträder. Detta ger en möjlighet att bestämma sannolikhetsfördelningar för de viktiga systemvariablerna. Man behöver även samla data om systemet för att senare, vid valideringen, kunna göra ordentliga fallstudier.
5) **Kodning** - processen som översätter beskrivningen av modellen till ett datorprogram som kan exekveras på en tillgänglig dator.
6) **Verifiering** - när man förvissar sig om att programmet fungerar på ett tillfredställande sätt enligt den modell som man har byggt upp.
7) **Validering** - när man förvissar sig om att programmet (och den modell man har skapat) överenstämmer med det ursprungliga systemet. Detta sker vanligen genom att simulera ett händelseförlopp som redan är känt, en fallstudie, genom tidigare datainsamling.
8) **Experimentutformning** - när man bestämmer sig på vilket sätt man ska variera de fria variablerna i modellen så att man får fram den utdata, i form av statistik, som man vill ha. Det innebär vanligen att man varierar invariablers värde, de fördelningsfunktioner som förekommer samt simuleringens längd.
9) **Produktionskörning och analys** - körning av programmet med de olika förändrningar som bestämts i steg 8, insamling av data, samt inte minst, beslutsprocessen om ytterligare simuleringar (programkörningar) behövs.
10) **Simuleringsrapportering** - dokumentation av programmet, beskrivning av modellen och rapport om resultatet av körningarna så att man får underlag nog att dra slutsater om det simulerade systemet.

Vid händelsestyrd simulering låter man simuleringen styras av de olika händelser som förekommer i modellen. En händelse definieras att inträffa vid en viss tidpunkt och, beroende på typen av händelse, förändra modellen. Alla förändringar i modellen sker vid en händelse på så sätt att händelsen startar en aktivitet som ändrar variabler i modellen och ev. skapar nya händelser vid andra tidpunkter. Alla händelser sorteras i kronologisk ordning i en s.k. händelselista.

Ett steg i en händelsestyrd simulering kan beskrivas i följande fem steg:

A) Tag första händelsen från händelselistan, dvs den vars tid är närmast den tid som systemklockan nu visar.
B) Flytta fram systemklockan till denna tidpunkt.
C) Utför de aktiviteter som denna händelse genererar. Posta ev. nya händelser på händelselistan. (Detta innebär egentligen att man skapar nya händelser, förser dem med de attribut som behövs, ger dem tidpunkter och sorterar in dem på rätt plats i händelselistan. Ingen nyskapad händelse har en tidpunkt med en tid som är mindre än systemklockans aktuella tid)
D) Samla statistik beroende på de förändringar i modellen som händelsen ger och vid vilken tidpunkt detta skedde. Väntetider är ofta intressant i t.ex. modeller där köer förekommer.
E) Börja om från punkt A igen.

Vid simulering använder man sig, som tidigare nämnts, av en hel del statistik. Problem som vanligen uppkommer är hur man ska beräkna de olika slumptalen som ska följa en given fördelning och hur man ska samla in data. Man måste dessutom på ett tidigt stadium bestämma sig för om det är en stokastisk studie eller en medelvärdesbild av systemet. Naturligtvis vill man inte att beräkningen av all statistik och generering av slumptal ska ta mycket exekveringstid från simuleringen. Det finns därför flera olika metoder att dels generera slumptal, dels beräkna statistik, allt för att spara tid.

För att exemplifiera vad som menas med en diskret händelsesimulering ges här ett exempel på en simulering av en maskin som är antingen igång (uppe) eller avstängd (nere). Vid varje tidsenhet som maskinen är uppe tillverkas en del. Vid starten av simuleringen genereras två händelser, en "uppe" vid tid 0, och en "stopp" vid tid 20. Sedan startas simuleringen. Första händelsen på händelselistan är en "uppe", varför denna processas. Denna skapar en "nere" om 5 tidsenheter (0+5=5) vilken sorteras in på händelselistan (överst). Nästa händelse i tur är just denna "nere" vid tid 5, som ökar antalet skapade delar med 5, samt skapar en "uppe" om 2 tidsenheter (5+2=7). På så sätt fortsätter simuleringen tills dess att händelsen "stopp" ligger överst på händelselistan och programmet bryts.

Tid	Status	Händelselistan vid tidpunkten	Tillverkade delar
0	uppe	Nästa fel vid 0+5 Stopp vid 20	0
5	nere	Nästa reparation vid 5+2 Stopp vid 20	5
7	uppe	Nästa fel vid 7+7 Stopp vid 20	5
14	nere	Nästa reparation vid 14+2 Stopp vid 20	12
16	uppe	Stopp vid 20 Nästa fel vid 16+7	12
20	uppe	Nästa fel vid 23	16

Fig. 1: Exempel på hantering av händelselista vid diskret händelsesimulering.

3. Parallell händelsestyrd simulering

Med parallell händelsestyrd simulering menar man att man kör en händelsestyrd simulering på en dator med ett flertal (>1) processorer. Att man har inriktat sig på just parallell händelsestyrd simulering beror på att simuleringar vanligtvis kräver enorma exekveringstider för att ge bra resultat (resultat som ligger nära funktionen hos det simulerade systemet). Man kan därför få kortare reell exekveringstid, men vanligtvis längre effektiv exekveringstid, om man delar upp simuleringen på flera processorer. Det optimala vore om den reella exekveringstiden var omvänt proportionell mot antalet processorer. Detta är tyvärr inte fallet p.g.a. att ju fler parallella processorer man använder, desto mer exekveringstid tar synkroniseringen, d.v.s. hanteringen av parallellismen.

Parallell händelsestyrd simulering är ett programmässigt mycket komplicerat problem. Man tittar speciellt på asynkrona system där händelsena inte synkroniseras av en global systemklocka, utan kan inträffa vid vilken tidpunkt som helst. I dessa system inträffar därför mycket få händelser vid samma tidpunkt, ofta färre än antalet tillgängliga processorer, varför man inte kan utnyttja datorn optimalt. Målet är därför att på något sätt tillåta samtidig exekvering, dvs hantering, av flera händelser som inträffar vid olika tidpunkter. Vi kommer se att detta introducerar flera intressanta synkroniseringsproblem vilket är själva teorin bakom parallell händelsestyrd simulering.

Tre olika datastrukturer är av störst betydelse:

1) Modellens statusvariabler som beskriver i vilken status modellen för tillfället befinner sig i.
2) Händelselistan som innehåller alla ej behandlade händelser som har skapats i modellen.
3) Klockan som talar om hur långt simuleringen har framskridit.

I den metod att hantera händelsestyrd simulering som beskrivits ovan är det mycket viktigt att hela tiden ta den händelse från händelselistan med lägst tidsangivelse (E_{min}). Om vi t.ex. tar en annan händelse med en större tidsangivelse, säg E_x, skulle det vara möjligt att E_x ändrar på statusvariabler som E_{min} använder sig av. På samma sätt kanske E_{min} egentligen skulle göra ändringar av statusvariabler som E_x använder sig av, vilket vid detta exempel aldrig skedde.

För att förenkla problemet, eller rättare sagt specialisera det, identifierar man i systemet olika fysiska processer. Dessa kan vara t.ex. kön till tvätten och själva tvätten i en modell av en enkel biltvätt. Om man vid denna biltvätt har två tvättar, kan man ha en köprocess samt två tvättprocesser, alltså tre olika fysiska processer. För varje fysisk process har man i modellen en logisk process (LP). Alla logiska processer har egna statusvariabler som beskriver motsvarande process status samt en lokal klocka. Alla logiska processer kommunicerar med varandra genom att skicka händelser (H), med tidsangivelse, mellan varandra. En logisk process kan naturligtvis även skicka en händelse till sig själv.

Men problem kan fortfarande uppstå. Antag att vi har två händelser, H_1 i den logiska processen LP_1 med tidsangivelse 10, och H_2 i den logiska processen LP_2 med tidsangivelse 20. Om H_1 postar en ny händelse, H_3, med en tidsangivelse mindre än 20 till logiska processen LP_2, så skulle H_3 kunna påverka statusvariabler som H_2 är beroende av. H_3 måste alltså behandlas av LP_2 innan H_2 behandlas för att händelsena ska hanteras i tidsordning.

Det diskuteras även andra sätt att utnyttja parallella processorer. Ett par varianter är:

1) Parallelliserande kompilatorer - att ha en kompilator som hittar delar av koden som kan exekveras parallellt på flera processorer. Denna metod är transparent för användaren och kan direkt användas på den stora mängd programvara för sekvensiell simulering som existerar. Detta ignorerar den parallellism som finns inbyggd i det ursprungliga problemet och därför utnyttjas endast en liten del av den möjliga parallellismen.

2) Distribuerade experiment - att på varje processor köra en separat simulering med unika indata. Detta snabbar upp simuleringstiden på en dator med N processorer med N. Men ibland är det inte bäst att köra många simuleringar. Det kanske är bättre att köra en längre simulering, och då använder man bara en processor. Parallellismen blir noll. Dessutom kanske det blir minnesproblem, då varje processor måste i sitt egna minne ha plats för en hel simulering.

3) Distribuerat språk - att dela upp uppgifterna inom hela simuleringen på olika specialicerade processorer. Man kan ha slumptalsgenereringen på en processor, statistikinsamlingen på en annan, in- och utmatning på en tredje o.s.v. På detta sätt har man kunnat uppnå en hel del bra resultat, men effektivitetshöjningen har ändock varit relativt låg jämfört med andra metoder.

4) Distibuerade händelser - att ha en global händelselista och exekvera två eller flera händelser parallellt endast om de anses "säkra", dvs inte kan komma att påverka varandra. För att klara av detta måste man ha en mycket klar bild innan simuleringen startar över vilka händelser som kan anses vara säkra och kan exekveras parallellt. Detta är en bra metod så länge som antalet processorer är litet.

5) Distribuerade delar av modellen - att identifiera olika delar av modellen som var och en för sig fungerar som en enhet, den tidigare beskrivna processen. Det visar sig att detta verkligen är den lättaste vägen att arbeta, då en uppdelning i olika logiska processer vanligtvis verkligen motsvaras av parallellt arbetande processer i det simulerade systemet.

Ett system med distribuerade delar av modellen måste synkroniseras på något sätt för att garantera att alla statusvariabler ändras i rätt ordning. Man kan synkronisera de olika delarna på flera olika sätt.

A) Konservativa metoder

I konservativa metoder ser man till att ingen händelse behandlas i en logisk process om man inte är säker på att ingen annan händelse med en tidigare tidsstämpel kan komma senare till samma logiska process. Man måste dessutom innan simuleringen startar veta exakt mellan vilka logiska processer det kan komma att skickas händelser.

B) Optimistiska metoder

I optimistiska metoder låter man simuleringen fortsätta ända tills dess att man upptäcker att det kommer en händelse H_m med en tidsstämpel T_n som är mindre än tidstämplarna T_{n+1} - T_m på de händelser H_n - H_{m-1} som redan har behandlats. Man har alltså fått ett tillfälle där en händelse kan komma att påverka tidigare händelser vilket inte är tillåtet. För att klara sig undan denna situation har man en rollback-mekanism som dels ogör alla händelser H_n - H_{m-1} (bakåt i tiden till tiden T_n) och dels skickar antihändelser till de processer som har fått dessa händelser så att de i sin tur får göra rollback. Efter detta kan man processa händelsen H_m med tidsstämpeln T_n, och sedan händelsena H_n - H_{m-1} på samma sätt som tidigare, men med annan status i processen.

4. Konservativa CM-metoden

I denna metod krävs att modellen har ett fixt, ändligt antal logiska processer, LP, och likaså ett ändligt antal identifierade riktade kanaler som binder ihop par av logiska processer. Händelser kan propagera mellan logiska processer endast via en kanal och endast i den riktning som är definierad. Det finns inga dubbelriktade kanaler utan dessa uttrycks som två motriktade kanaler. Varje logisk process kan exekvera sekventiell kod samt de två speciella kommandona send och recieve. Send innebär att en logisk process skickar ut ett meddelande på en kanal samt recieve att den vill ta emot ett meddelande från en (annan) kanal.

Ett kronologiskt krav finns också. Om en logisk process LP_i skickar en händelse H_i med tiden T_i till den logiska processen LP_j på en kanal K_{ij} kommer alla framtida händelser H_{i+1} - H_∞ ha tider T_{i+1} - T_∞ som alla är senare än T_i. Vi säger att tidsstämpeln Q_{ij} på kanalen K_{ij} är lika med tidsstämpeln T_i på den senaste händelsen H_i som passerade på den kanalen. $T_i = Q_{ij}$. Därmed kan en LP behandla alla väntande händelser i den lokala händelselistan som har en tidsstämpel tidigare än $\min(Q_{nm})$.

En mängd logiska processer, D, sägs vara i deadlock vid något tillfälle om:

1) Varje LP i D antingen väntat på att ta emot en händelse eller har terminerat.
2) Minst en LP i D väntar på att ta emot en händelse.
3) Att någon LP_i i D väntar på att ta emot en händelse från LP_j som också ligger i D och det inte finns några händelser på kanalen mellan LP_j och LP_i.

Det visar sig att ingen LP i D kan fortsätta sina beräkningar p.g.a. att de väntar på varandra.

Ett sätt att undvika deadlock är att skicka s.k. nullmeddelanden i modellen som inte återfinns i det simulerade systemet. En logisk process LP_i som ska skicka en händelse till den logiska processen LP_k med tidsstämpeln T_i skickar även nullmeddelanden till alla andra logiska processer som den har kontakt med. Alla utgående kanalers tidsstämplar kommer därmed ändras till T_i.

Meddelandet innehåller alltså en tidsstämpel T_i och är till för att LP_i ska meddela LP_j att inga framtida händelser kommer ha en tidsstämpel med tidigare tid än T_i.

Antag att man tittar på den graf som representeras av kanalerna K_{00} - K_{pq} som kanter samt de logiska processerna LP_0 till LP_r som noder. I denna graf kan man hitta cykler. En deadlock måste uppstå i en sådan cykel. Om man använder sig av nullmeddelanden och sätter kravet att inte någon cykel har total behandlingstid av en händelse som är lika med noll kommer ingen deadlock att uppstå.

Med total behandlingstid menas summan av minimala behandlingstiden i såväl varje kanal som varje logisk process. Även en kanal kan nämligen ha en behandlingstid av en händelse och denna tid kan variera. Om en simulering ska sluta vid tiden T och en logisk process inte har några fler händelser kvar att skicka och inte heller kommer ha det, ska ett nullmeddelande skickas med tidsstämpeln T. Därmed kommer kanalens tidsstämpel ändras till T.

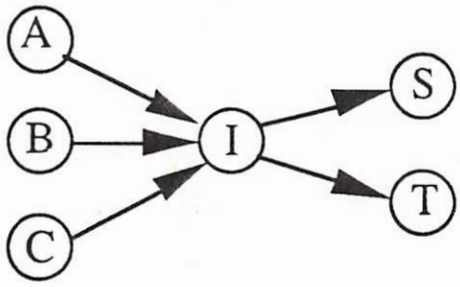

Fig. 2: Denna figur visar den struktur av logiska processer som senare kommer exemplifiera skickandet av nullmeddelanden. Bokstaven A betecknar den logiska processen LP_a, pilen till I den riktade kanalen K_{ai} o.s.v.

Antag att den logiska processen LP_i har ingående kanaler från LP_a, LP_b och LP_c. Minsta betjäningstiden hos LP_i är 4. Utgående kanaler är Q_{is} och Q_{it} (Se figur 2). Q_{ai}=0, Q_{bi}=2 och Q_{ci}=1. Dessa tidsstämplar kommer från antingen händelser eller nullmeddelanden som har skickats på kanalerna K_{ai}, K_{bi} eller K_{ci}. Den minsta tiden som här förekommer är Q_{ai} som är 0. Om meddelandet från LP_a var ett nullmeddelande skickar LP_i nullmeddelande på kanalerna K_{is} och K_{it} där tidsstämpeln är 4 (0+4). Därmed kommer Q_{is} och Q_{it} bli 4. De andra händelsena som kom från LP_b och LP_c läggs upp på händelselistan vid tidpunkterna 2 resp 1.

Steg	LP_i Send LP_i	LP_b Send LP_i	LP_c Send LP_i	LP_i HL	LP_i Send LP_s	LP_i Send LP_t
1	(0,n)	($2,H_2$)	($1,H_1$)			
2				(0,n) ($1,H_1$) ($2,H_2$)		
3				($1,H_1$) ($2,H_1$)	(4,n)	(4,n)
4	($1,H_3$)	(3,n)	(2,n)	($1,H_1$) ($2,H_2$)		
5				($1,H_1$) ($1,H_3$) ($2,H_2$)		
6				($2,H_2$)	($5,H_5$)	(5,n)
7	(T,n)	(T,n)	(T,n)		(7,n)	($7,H_6$)
8					(T,n)	(T,n)

Fig. 3: Detta diagram visar vilka meddelanden som skickas vid varje tidssteg i modellen som finns i Figur 2. LP_a, Send LP_i betecknar händelse eller nullmeddelande som LP_a skickar till LP_i. LP_i, HL betecknar den lokala händelselistan i den logiska processen LP_i vid varje tidpunkt.

Diagramet i fihur 3 visar vilka nullmeddelanden som LP_a skickar på den kanal som inte den genererade händelsen ska gå på. Tidsstämpeln blir $\min(Q_{ai}, Q_{bi}, Q_{ci}) + B_i$ där B_i betecknar den minsta betjäningstiden som kan förekomma vid LP_i.

Att nullmeddelanden ser till att förhindra deadlock i en cykel med icke-noll betjäningstid kan visas enkelt. Antag att vi har en cykel av logiska processer LP_0, ...,LP_n. Mellan processerna går riktade kanaler K_{01}, K_{12}, ..., K_{n0}. Definiera dessutom betjäningstiden vid varje logisk process som B_0, ..., B_n. Vid början av simuleringen har LP_0 en händelse vid tiden 1, alla LP har lokal tid 0. Eftersom alla Q_{01}, ...,Q_{n0} därmed är 0, kan ingen händelse skickas (händelsen vid LP_0 kan inte heller utföras). Då skickar alla logiska processer ett nullmeddelande. Från LP_0 kommer t.ex. $N(B_0)$ skickas. På liknande sätt kommer det från LP_k skickas nullmeddelande $N(B_k)$. Därmed kan den lokala tiden på LP_{k+1} flyttas fram till B_k. Nästa nullmeddelande från LP_{k+1} till LP_{k+2} blir $N(B_k+B_{k+1})$. Efter att nullmeddelanden har passerat ett helt varv kommer LP_k få meddelandet $N(B_k+B_{k+1}+...+B_n+B_0+...+B_{k-1})$. Då någon av B_0, ...,B_n är strikt större än noll, kommer den lokala klockan kunna flyttas fram till detta värde. Tiden är alltså strikt växande och deadlock har inte uppstått.

5. Jämförelse med optimistiska metoder

Vid konservativa metoder synkroniserar man hela tiden de lokala tidpunkterna i varje logisk process med hjälp av extra meddelanden. I CM-metoden s.k. nullmeddelanden. Detta är en extra arbetsbörda, och en hel del exekveringstid går åt för detta, varvid simuleringen får en allt mindre del av exekveringstiden.

Vid optimistiska metoder slipper man denna synkronisering så länge allt går bra. Men när man upptäcker antingen ett deadlock eller att en händelse har en för tidig tidsstämpel måste man avbryta stora delar av modellen och antingen låsa upp det deadlock som uppstått eller ogöra en del händelser som redan har behandlats. Detta kräver i bägge fallen mycket arbete.

Flera undersökningar har gjorts där man jämför den konservativa CM-algoritmen med någon optimistisk metod. I en jämförelse på en BBN Butterfly™ med 17 processorer fick man vid den konservativa metoden ungefär dubbelt så bra speedup (jämförelse mellan resultatet på det parallella systemet och en motsvarande simulering på en enprocessorssimulering) om man använde konservativa CM-metoden istället för en optimistisk algoritm.

Resultatet beror på flera olika faktorer:

- Den minimala betjäningstiden (lookahead ratio) - ju längre minimal betjäningstid desto bättre fungerar CM-algoritmen.
- Antalet logiska processer - ju fler processer, desto mer parallellism, men också mer overhead vid båda metoderna.
- Antalet händelser - ju fler händelser, desto mindre del av arbetet är hantering av nullmeddelanden vid CM-metoden - ju fler händelser desto mer deadlock och därmed mer extraarbete vid optimistiska metoder.

Ur detta kan man dra flera slutsatser. Den viktigaste är kanske att återigen poängtera att varken optimistiska- eller konservativa metoder passar för simuleringar där man inte känner de olika händelser som kan inträffa. De passar t.ex. inte för simuleringar av beslutsprocesser. Inte heller passar de bra för simuleringar av förlopp som inte i sin uppbyggnad kan parallelliseras i flera självständiga logiska processer. Ännu svårare är det att få parallella händelsestyrda simuleringar att fungera för simulering av dynamiska system såsom slagfält.

Däremot passar de utmärkt för simulering av händelser i processindustrin, av datorer och datorsystem, av kommunikationsnät, av processorer m.m. Av dessa, optimistisk resp. konservativ metod visar det sig att den konservativa är bäst ur effektivitetssynpunkt. Den blir bättre och bättre ju längre minimala betjäningstider det är och bättre och bättre jämfört med den optimistiska metoden ju fler händelser som skickas mellan de olika processerna.

I simuleringar som innehåller låga värden på minimal betjäningstid eller låga möjligheter till parallellisering (i form av många riktade kanaler mellan logiska processer) blir dock allt extraarbete mycket stort, till och med i ett system som använder konservativa metoder, att andra simuleringsmetoder än distribuerad diskret händelsesimulering rekommenderas.