# Beräkna framtiden, idag!

Time Warp-algoritmen

P E T T E R  E R I K S S O N


KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

# Beräkna framtiden, idag!

Time Warp-algoritmen

P E T T E R  E R I K S S O N

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Cristian Bogdan
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/eriksson_petter_K10027.pdf

Kungliga tekniska högskolan
*Skolan för datavetenskap och kommunikation*

**KTH** CSC
100 44 Stockholm

URL: www.kth.se/csc

## Abstract

This paper is about the time warp algorithm. The algorithm will be explained so that the reader gets an idea of how the algorithm works.

With this algorithm come non-trivial pitfalls that are easy to fall into, if one is not careful. The algorithms performance can be greatly compromised, if these issues are not taken care of.

The paper also includes examples of how to solve these issues and applications using the algorithm today.

# Innehållsförteckning

# Inledning

Processorers klockfrekvenser blir inte längre högre och högre, utan dagens processorer får fler och fler kärnor per processor [2]. Beräkningar kommer inte längre kunna gå snabbare om man inte börjar parallellisera uträkningarna. Om vi vill spela mer avancerade spel i framtiden, kommer utvecklare att måsta parallellisera uträkningarna som sker i spelen.

Time Warp är en algoritm som parallellt kan räkna ut problem som är helt beroende av händelser kring en tidslinje. En komplex algoritm som sammanfattningsvis går ut på att gissa, kontrollera, göra om och göra rätt.

## Bakgrund

Time Warp algoritmen skapades i mitten på 1980-talet av David Jefferson [1]. På den tiden trodde de flesta forskare att det var omöjligt att implementera en asynkron miljö som är baserad på att man gör rollback vid lägen då ett fel inträffar, eller att det var oöverkomligt kostsamt i datatid att lösa problem med en Time Warp algoritm.

## State of the art

Även om det är en gammal algoritm och ett gammalt tänkesätt att lösa problemet, så lär universitet, Kungliga Tekniska Högskolan, fortfarande ut Time Warp i dagsläget [5]. Att räkna ut och lösa simulationsproblemet med hjälp av att dela upp problemet över flera processorer och sedan räkna de olika delarna i simulationen parallellt, är fortfarande modernt. Det är även något som ser ut att fortsätta vara modernt i framtiden [2, 5].

Idag används Time Warp t.ex. till simuleringar av massvis med kroppar. Vid sådana simulationer är det mycket effektivare att använda en Time Warp algoritm med virtuell tid, istället för att använda en algoritm där kropparna alltid måste vara synkroniserade [4]. Time Warp används även till nätverksbaserade dator- och tvspel, där det är viktigt att det inte finns synliga nätverksfördröjningar. Med hjälp av Time Warp kan man, på ett effektivt sätt, se till att spelarnas handlingar är så korrekta som möjligt, utan att låta nätverkets fördröjningar vara inblandade i uträkningarna [7].

David W. Bauer Jr. är en av dem som har lyckats att få stark prestandaökning med Time Warp på multikärnade processorer. David använde sig av IBM:s superdator Blue Gene, som har upp till 131 072 processorer [8]. När de simulerade PHOLD modellen fick de en stark skalnings topp vid 65,536 processorer, och vid simuleringen av TLM modellen fick de en svag skalning mellan 2 000 och 10 000 processorer. Tidigare har man bara lyckats få skalning upp till 25 processorer, på samma modell. Detta visar på att Time Warp går att effektivt parallellisera. De beskriver även i sin rapport att de tror att de kan få en högre topp på PHOLD simuleringen.

## Syfte

Algoritmen är bra på att simulera problem där de objekt som ska simuleras är beroende av andra objekt i simulationen. Exempel på detta är nätverks spel där spelare skjuter på varandra[7]. Där skott är beroende på var den andra spelaren befinner sig, och simulationer av bollar som åker ner för en backe och krockar med varandra [4]. Time Warp simulerar detta bra eftersom att Time Warps beräkningar är parallelliserbara[1, 5], då man gissar på sannolika händelser och återställer dessa gissningar om de råkar vara felaktiga.

Petter Eriksson
880209 – 8676



Time Warp erbjuder mycket prestandaökning om den är implementera rätt. Men att gissa på händelser och att sedan stega tillbaka till ett korrekt tillstånd kan ställa till med massor med problem.

I den här rapporten förklaras Time Warp algoritmen, så att läsaren ska få en uppfattning av om hur algoritmen fungerar. Fokus ligger dock på algoritmens problem och fallgropar som implementerare av algoritmen bör känna till. Vi går igenom förslag på hur man ska lösa dessa problem och tittar även på några applikationer som är baserade på idén.

# Algoritmen

Med Time Warp algoritmen kan man beräkna händelsebaserade problem, där varje händelse beror på tidigare händelser. Simulering av kroppar med kollision är ett exempel på ett sådant problem [4]. Problemet är att räkna var godtyckligt formade kroppar befinner sig, efter att ha rört sig med någon hastighet i någon riktning. I de fallen där man har en ganska stor värld relativt till hur många kroppar som rör sig, spenderar kropparna en väldigt liten tid att krocka med varandra. Istället för att stega fram varje kropp och undersöka om de krockar med något, så gissar man på en sannolik händelse – t.ex. att de fortsätter att röra sig i samma riktning – och kontrollerar sedan om denna gissning var rätt. Det är inte nödvändigt att kontrollera gissningen direkt, utan den som har gissat på en händelse kan anta att gissningen är giltig, och gissa flera gånger utan att kontrollera att de är sanna. På så sätt spenderar man mindre av sin tid på att kontrollera den osannolika händelsen att kroppar krockar, vilket också är väldigt dyrt att kontrollera.

Låt oss förklara detta mer detaljerat.

## Förklaring

Algoritmen består huvudsakligen av tre bitar [1, 4, 5, 8].

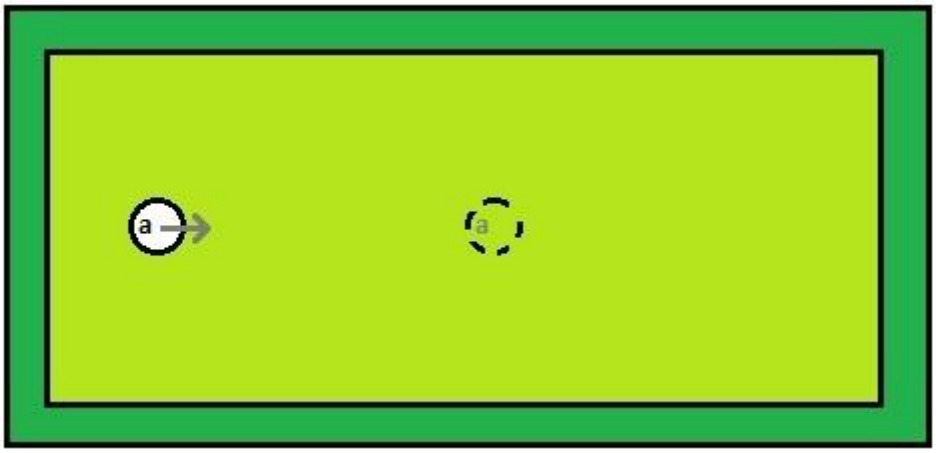
- Logiska processer som utför beräkningar och kommunicerar med andra logiska processer.
- Meddelanden som har olika betydelser beroende på vilken typ av meddelande det är.
- Rollbacks som återställer en logisk process till ett stabilt tillstånd.

För att göra de olika delarna lättare att förstå för läsaren, ingår förklaringarna av bitarna i ett exempel av en biljardbordsstöt.

### Logiska processer

För att Time Warp ska kunna simulera en simulationsmodell ska alla fysiska föremål i modellen, kunna översättas till en logisk process eller (LP) [1]. En logisk process skulle i biljardbordsstöten vara en av bollarna på ett biljardbord. En sådan logisk process skulle kunna hålla information om bollens storlek, hastighet, riktning och position.

För att avancera i simuleringen ska de logiska processerna kunna gissa sig till vad som händer med dem. En sannolik händelse för en biljardboll är fortsätta i samma riktning med bollens hastighet.




Figur 1: En logisk process representerar $boll_a$ som har en viss hastighet (grå pil). Den streckade cirkeln är en gissning på var bollen bör befinna sig efter någon tid.

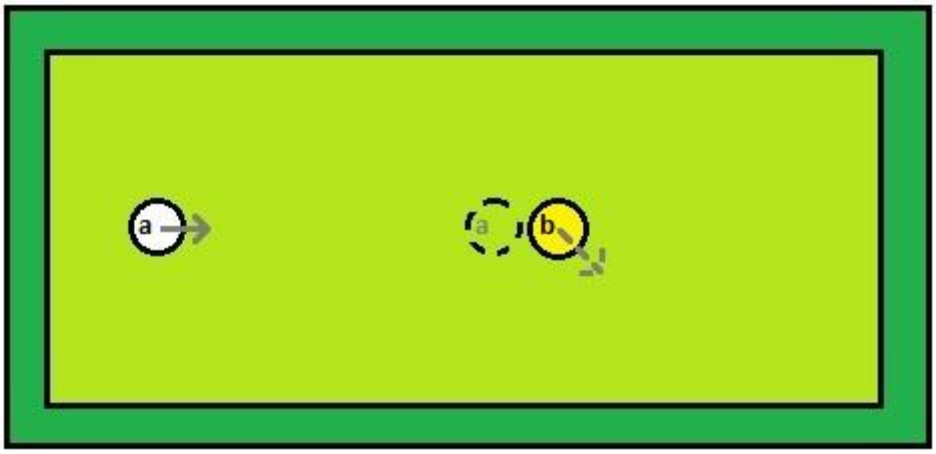
## Meddelanden

Meddelanden innehåller information om interaktioner mellan LP:s. Varje LP kan bara kommunicera med andra LP:s via tidsstämplade meddelanden.

Varje gång det skickas ett meddelande, skapas också ett antimeddelande, på logiska processen som skickade meddelandets output kö [1,4]. Antimeddelandet är sedan redo att skickas till den logiska process som har mottagit meddelandet. Detta sker eftersom att meddelandet var ett antagande och kan därför vara ett felaktigt meddelande.

Ett antimeddelande innehåller beskrivning för att göra en händelseutförelse, baklänges. Det vill säga att om man utför ett meddelande och dess motsvarande antimeddelande – direkt efter varandra - skulle den logiska processen hamna i samma tillstånd som den befann sig i innan den utförde något av meddelandena. För ett meddelande, m, och ett antimeddelande, -m, gäller att -(-m) = m [1].

Vi fortsätter på biljardstötexemplet. Säg att $boll_a$ gissar att den kolliderar med $boll_b$. Ett meddelande kommer då att skickas från $boll_a$ till $boll_b$ som har tidstämpel, t, och data beskrivande hur de krockade. Det läggs också ett antimeddelande på $boll_a$ output kö, som har samma tidstämpel, t, och beskriver hur man ska ångra krocken [Figur 2].

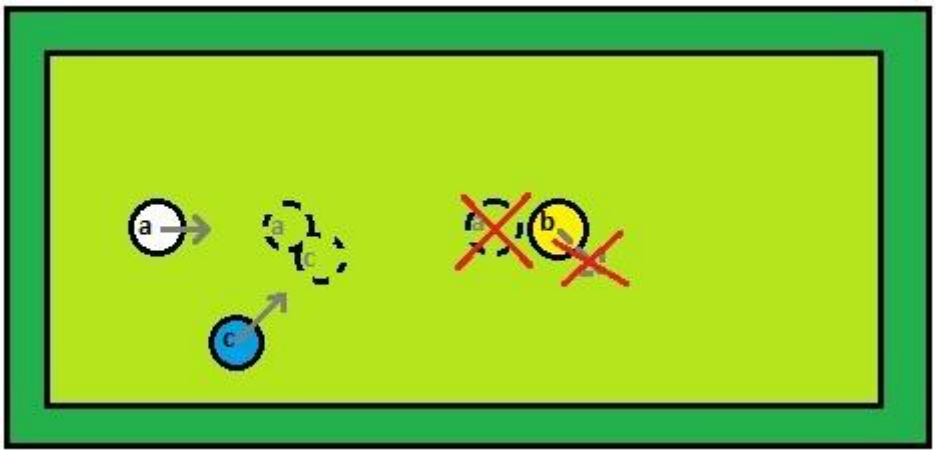



Figur 2: Den logiska processen som representerar $boll_a$ – vi kallar denna för $LP_a$ - räknar ut att den kommer att kollidera med $boll_b$. $LP_a$ skickar ett tidstämplat meddelande med information om krocken till $LP_b$. $LP_b$ kan sedan göra sannolika gissningar på hur krocken påverkar $boll_b$.

## Rollback

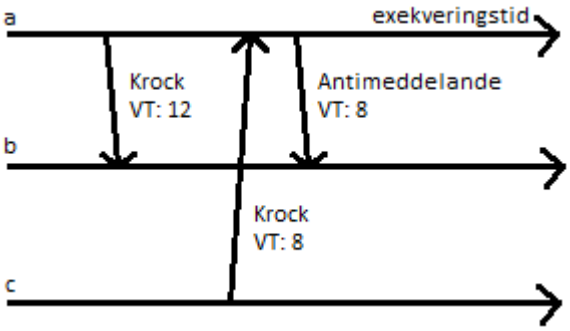
Med rollback menar man att en LP återgår till ett stabilt tillstånd. En rollback inträffar när en LP har gjort en beräkning på felaktig indata [1,5]. Rollbacken kan ske antingen genom att man utför antimeddelanden eller att ett tidigare sparat tillstånd laddas.

En rollback skulle kunna ske om $boll_a$ får ett meddelande att den har krockat med $boll_c$ vid ett tillfälle tidigare än tid, t, i förra exemplet. $Boll_a$ skickar ut sitt antimeddelande till $boll_b$, så att $boll_b$ kan komma tillbaka till ett tidigare tillstånd. Gör både $boll_a$ och $boll_b$ också rollback till den nya tiden då $boll_c$ kolliderade [Figur 3].




Figur 3: $Boll_c$ skapar en rollback för $boll_a$ och $boll_b$.

Varje LP kan som nämnt innan gissa vad som händer dem. Detta kan de göra oberoende av hur långt de andra logiska processerna har hunnit beräkna i simuleringen. Därför kan olika LP:s ha beräknat olika långt i simuleringstid, som även kallas för virtuell tid (VT), och ordningen som de logiska processerna gör sina beräkningar varierar. Figur 4 visar hur $LP_a$ har hunnit beräkna till VT:12 och säger att den krockar med $LP_b$. Senare får $LP_a$ ett meddelande som säger att det skedde en krocka innan VT:12. Detta genererar en rollback och antimeddelanden skickas.




Figur 4: Exekverings ordning i biljardbollsexemplet. Varje horisontellpil representerar varsin LP. Pilarna som går mellan två LP:s är ett meddelande.

## Mer om algoritmen

För att få ut så mycket parallellism som möjligt ur Time Warp algoritmen, vill man vara optimistisk och skicka meddelandena mellan LP:s asynkront [8]. Med att vara optimistisk menas det att ge logiska processerna frihet att gissa och beräkna långt in i den virtuella simuleringstiden. Optimismen kommer ifrån att det antas att de logiska processerna aldrig eller väldigt sällan beräknar fel. Om kommunikationen mellan dem också sker asynkront, kan de logiska processerna fortsätta sin simulering utan att vänta på att mottagaren av meddelanden ger en bekräftelse.

Med hjälp av optimism och asynkron kommunikation låter man alltså logiska processer beräkna så mycket som möjligt, oberoende av någon annan. Dessa LP:s kan man också distribuera över flera processorkärnor och få den härliga multikärniga skalning som framtiden vill ha [2]. Men att vara optimistisk kräver också mycket minne, samt att meddelanden tar ett tag att skicka mellan processer vilket skapar andra intressanta problem. Mer om risker och problem i just ”Risker och problem” avsnittet.

## En liknande algoritm

En algoritm som Jefferson återkommer till upprepande är Chandy-Misra [1]. Chandy-Misra har – likt Time Warp – kravet på att varje fysiskt föremål ska kunna beskrivas som en logisk process, samt att processerna bara kan kommunicera med andra processer via tidsstämplade meddelanden[1, 12]. Den här liknande algoritmen kan vara bättre på att räkna ut ett beteende på ett statiskt nätverk av interaktiva processer, där alla processer har ungefär lika stor trafik. Dock har Chandy-Misra ständiga problem med att det kan bli ett så kallat deadlock eller baklås. Ett deadlock är då två eller flera processer väntar på varandra och kan inte bli färdiga. Algoritmen har också restriktioner som inte är triviala [1]. Man kan till exempel inte alltid skicka ett meddelande från en godtycklig $LP_i$ till en annan $LP_j$, där $LP_i \neq LP_j$.

# Risker och problem

Jefferson skrev sin rapport [1] 1987. Redan då hade han frågeställningar som kom att en spela roll i framtiden, ironiskt nog. Han undrar i rapporten hur Time Warps prestanda minskar då det blir ont om minne. Några år senarer adresserade D.M. Nicol och X. Liu risken att skapa antimeddelanden som inte kan tydas av en LP i det tillstånd som den befinner sig i [3]. I början på 2000-talet beskrev R.M. Fujimoto problemet med rollbacks som i sin tur kan skapa en ny rollback [9]. Dessa tre besvär är beskrivna mer detaljerat i detta avsnitt.

## Resurser

När man använder sig av Jeffersons optimistiska tillvägagångssätt, ska man vara försiktig med att inte låta en LP springa iväg allt för långt med sina gissningar. Det tar minne att hålla kvar antimeddelanden så att det är möjligt att göra rollback till ett säkert tillstånd. Det bör finnas någon gräns på hur mycket en LP får göra på egen hand utan att gissningarna kontrolleras, för att sedan frigöra det minne som en LP har allokerat.

## Antimeddelanden

En av riskerna finns vid skapandet och skickandet av de antimeddelanden som Jefferson beskriver [1, 3]. Risken är att det antimeddelande som skapas när man gissar en händelse, kommer till den LP som ska hantera meddelandet, när processen befinner sig i ett sådant tillstånd att den inte kan tolka meddelandet på ett vettigt sätt.  I bästa fall kraschar programmet, men programmet kan också tolka meddelandet på ett sådant sätt att den skapar felaktig data som aldrig skulle kunna hända i verkligheten [3]. Detta kan vara katastrofalt för en simulering, då det kanske upptäcks senare och man kan inte säkert göra en rollback till ett säkert tillstånd, eller så upptäcks det aldrig så att simuleringen ger helt felaktiga data. Ett antimeddelande som inte går att tolka kan även skapa ett dödslås så att simuleringen aldrig blir klar.

## Domino rollbacks

Det är möjligt att algoritmen, i vanlig ordning, upptäcker att en LP har gjort ett felaktigt antagande eller gissning för att göra en rollback. Den rollback som har gjorts garanterar dock inte alltid att programmet eller den logiska processen, hamnar i ett säkert tillstånd. Eftersom rollbacken kan leda till ett osäkert tillstånd som i sin skapar ännu en rollback, finns det en risk att programmet hamnar i en oändlig loop med rollbacks [9].

Förslag på lösningar till dessa problem i anges nästa avsnitt.

# Lösningar

De problem som nämns i problem avsnittet och som Jefferson tänkte på i mitten av 80-talet, har man idag löst på olika sätt. Vi tittar på några exempel på hur man kan tackla problemen och hur man kan undvika riskerna med Time Warp.

## Global Virtual Time

Global Virtual Time – vilket förkortas GVT – definierar en den lägsta gränsen för den tid som de tidsstämplade meddelanden får innehålla [4, 5]. GVT är ett sätt att ha kontroll på hur långt fram i tiden man har försäkrat sig om att processerna har gjort korrekta simuleringar. Man kan inte göra rollback till en tid som är tidigare än GVT. Det kommer heller inte behövas, då man är garanterad att allting som har hänt före GVT tiden är korrekt.

Eftersom man kan försäkra sig att allting före GVT är korrekt, kan man utföra input/output operationer som är definitiva. Man kan också återta minne som innehåller sparade tillstånd som ligger före GVT, eftersom man ändå inte göra en rollback till ett sådant sparat tillstånd [5, 13].

GVT löser alltså minnesresursproblemet om det körs nog ofta. Att sätta den här lägre säkra gränsen kommer dock inte utan sitt pris. Att köra kontrolleringsalgoritmen är processorkrävande och man vill helst göra det så lite som möjligt, för att ge processorerna så mycket tid som möjligt att simulera.

## Synkronisering av meddelanden

En LP kan inte kontrollera giltigheten av ett mottaget meddelande eller antimeddelande[3]. Det finns kända teoretiska svårigheter när man ska kontrollera ett meddelande som ska exekveras, då det inte finns någon algoritm som kan veta om godtycklig indata till ett kodstycke terminerar inom ändligtid [6]. I värsta vall är det obestämbart att kontrollera om ett antimeddelande är korrekt. Det är heller inte trivialt för en implementerare av Time Warp, att förhindra felaktiga antimeddelanden från att skickas iväg ifrån en LP.

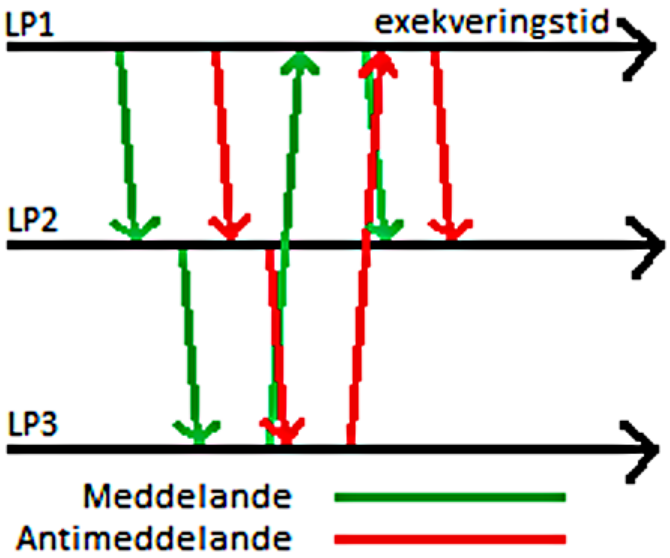
Istället för att kontrollera då antimeddelandet kommer till processen, kan man synkronisera meddelanden med mottagaren. Det vill säga att mottagaren av meddelandet skickar tillbaka en bekräftelse till avsändaren, som säger att meddelandet är mottaget och behandlat.

Genom att synkronisera med den nämnda metoden, är det också lättare för den som utvecklar mjukvaran att verifiera säkerheten i programmet. Synkroniserade meddelanden ökar säkerheten eftersom att det går att verifiera vilka meddelanden som går bra att processera och vilka meddelanden som fallerar [3].

Denna synkronisering har dock pris. För varje meddelande som sänds måste det skapas en bekräftelse. Det betyder att trafiken mellan processerna dubbleras. Synkroniseringen av meddelanden gör också att det blir parallellism i algoritmen, då de logiska processerna måste vänta på svar, istället för att fortsätta beräkna oberoende av andra LP:s.

## Proaktiv avbrytningsmekanism

För att undvika domino rollbacks och katastrofala tillstånd för sina LP:s, går det att implementera en proaktiv avbrytningsteknik som heter PTC (Plausible total clocks) [9]. Med en proaktiv avbrytningsteknik menas det att man gör en extra ansträngning för att avbryta en LP då det sker en rollback.




Figur 5: Ett katastrofalt tillstånd där antimeddelanden jagar svansen på vanliga meddelanden.

När det ska ske en rollback i en simulation, är det viktigt att alla logiska processer som har simulerat längre än den tidpunkten där felet upptäcktes, avbryts och gör rollback till denna tidpunkt. Om någon LP inte skulle avbrytas, kan den fortsätta skicka felaktig information vidare till en process som redan har återvänt till ett säkert tillstånd.

Avbrytningen går att tvinga fram genom att hålla en vektor med ett index för alla logiska processer i varje LP, som räknar upp när det sker en operation mellan två logiska processer. Med hjälp av denna vektor - som kallas ”vector counter” (VC) - kan man kontrollera hur många giltiga operationer som har gått igenom sedan det tidigaste säkra tillståndet. Med hjälp av denna VC och med tidstämplar, kan det ske säkra rollbacks, då det inte processeras meddelanden som kommer ifrån en LP med en VC som kan bevisas att innehålla felaktiga operationer.

Vektorn som räknar antalet operationer växer linjärt mot antalet logiska processer i simulationen. En uppdaterad vektor måste också ingå i varje meddelande som skickas mellan de logiska processerna. Detta medför att det krävs mera minne per logisk process, samt att trafiken ökar.

# Implementationer idag

Idag används Time Warp till simulationer av stora system på superdatorer [4, 8] och till att lösa synkroniseringsproblem i nätverksspel [7, 10].

## Nätverksspel

Time Warp är bra att använda i snabba spel som kräver att man alltid har full kontroll över sin spelpjäs. Exempel på ett sådant spel är tankskjutarspelet som S. Tolic and H. Hlavacs skapade [7].

Det kan behövas andra metoder än bara Time Warp för att göra spelet tillräckligt effektivt och konsekvent för att kunna spelas. Med hjälp av att simulera en lokal fördröjning för varje klient, kan man se till att de meddelanden som skickas över TCP hamnar i ordning. Denna lokala fördröjning förhindrar dock inte alla inkonsekvenser, så därför kallar man på Time Warp när det behövs [7, 10].

## MERL

Time Warp används också för att, räkna ut hur en modell som består av flera kroppar interagerar med varandra när de kolliderar [4].

Time Warp har egenskapen att slippa kolla kollisioner synkront bland alla kroppar. När kropparna också har mycket utrymme att röra sig på, är sannolikheten stor att gissningarna blir korrekta och att det blir få rollbacks. Denna egenskap och fallet med en stor rymd relativt till volymen av kropparna, gör att Time Warp – med hjälp av GVT – är en effektiv algoritm för att simulera denna modell.

# Diskussion

Rapporten har skrivits med hänsyn till att Jefferson kan ha vinklat sin egen rapport om Time Warp till sin egen fördel. Hans teorier har dock verifierats av forskare och doktorer, vilket gör att Jeffersons rapport – som detta arbete är grundat på – blir mera trovärdig.

De lösningar som har föreslagits är bara några exempel på hur man kan göra besvären med Time Warp mindre. GVT, synkronisering av meddelanden och PCT ökar alla säkerheten på algoritmen, men de sänker också alla prestandan.

GVT är en välkänd metod som har nämnts i olika arbeten [4, 5, 13]. Denna metod ökar säkerheten och frigör minne, men det är väldigt dyrt i CPU-tid att köra GVT. Därför måste implementerare vara noga med att tänka igenom vilket värde som ska sättas på parametern som bestämmer hur ofta GVT ska köras.

Synkronisering av meddelanden är något som man helst vill undvika. Synkroniseringen skadar Time Warps viktigaste egenskap, att kunna utföra beräkningar parallellt. När en logisk process interagerar med en annan, måste denna LP vänta på svar. De logiska processerna ödslar CPU-tid på att vänta istället för att fortsätta utföra sina beräkningar oberoende av vad mottagaren gör. Denna metod öppnar också upp risken för dödslås, vilket är förödande för alla slags uträkningar.

PCT ökar trafiken i systemet och minnet per logisk process. Metoden ökar dessa två faktorer linjärt mot antalet logiska processer i systemet. Det är oklart hur mycket prestanda man betalar för den extra säkerheten som PCT erbjuder.

Time Warp algoritmen är en komplex algoritm att implementera. Det finns många icke triviala problem och oförutsedda händelser som kan uppstå under av en simulation som implementerar algoritmen. Nyligen har det skett forskning för att göra det säkrare att använda sig av algoritmen, och Time Warp användes för att slå rekord i antal event per sekund under en PHOLD simulering [8].

Parallellisering av beräkningar är framtiden. Intel tror att det dyker upp processorer med upp till 80st kärnor i en processor år 2012 [11]. Att simulera objekt i en virtuelltid där man gör rollback vid eventuella felaktiga antaganden för att skapa en verklig tidslinje, är uppenbarligen möjligt. Modifikationer av Time Warp används idag och framtiden ser bara ljusare ut för denna algoritm.

# Referenser

[1] D.Jefferson & al, "Time Warp Operating System", ACM Symp. on Operating Systems Principles, ACM, Austin TX., (1987), http://doi.acm.org/10.1145/41457.37508

[2] Vladimir Vlassov, ”Introduction. Parallel Programming Concepts, Models and Paradigms”, KTH/ICT/ECS, VT (2010), http://www.imit.kth.se/courses/ID1217/handouts/lec01.pdf

[3] D. M. Nicol, X. Liu, "The dark side of risk (what your mother never told you about Time Warp)", ACM SIGSIM Simulation Digest archive 27, 1, (1997), http://doi.acm.org/10.1145/268823.268920

[4] Brian Mirtic, "Timewarp ridgid body simulation", Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 193 - 200, (2000), http://doi.acm.org/10.1145/344779.344866

[5] – Rassul Ayani , “Parallel Discrete Event Simulation”, KTH/ICT (2009), http://www.imit.kth.se/courses/IV1200/lectures/L06.pdf

[6] – Walker, H. M, “*The Limits of Computing”*, Jones and Bartlett Publishers, Inc (1994)

[7] – Stefan Tolic, Helmut Hlavacs, “A Testbed for P2P Gaming Using Time Warp”, University of Vienna, Dept. of Distributed and Multimedia Systems, (2009)

[8] – David W. Bauer Jr, “Scalable Time Warp on Blue Gene Supercomputers”, High Performance Technologies, Inc., (2009)

[9] – Malolan Chetlur, Philip A. Wilsey, (2009) , “Causality information and proactive cancellation”, Concurrency Computat.: Pract. Exper. 2009; 21:1483–1503, http://portal.acm.org/citation.cfm?id=1568498.1568502

[10] – M. Roccetti , S. Ferretti, C.E. Palazzi, (2008), ”The Brave New World of Multiplayer Online Games: Synchronization Issues with Smart Solutions”, Dept. of Comput. Sci., Bologna Univ., Bologna, ISBN: 978-0-7695-3132-8, http://www.computer.org/portal/web/csdl/doi/10.1109/ISORC.2008.17

[11] – Multi core och multi thread är framtiden
av Niklas Andersson (2008-06-19)
http://www.idg.se/2.1085/1.168032/multi-core-och-multi-thread-ar-framtiden

[12] - Jari Porras, Jouni Ikonen, Jarmo Harju, (1998), “Applying a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network”, http://portal.acm.org/citation.cfm?id=278009.278031

[13] – Richard M. Fujimoto, Maria Hybinette, (1997), “Computing Global Virtual Time in Shared-Memory Multiprocessors”, Georgia Institute of Technology, http://portal.acm.org/citation.cfm?id=268404