

Verteilte Algorithmen

53732 characters in 7324 words on 1347 lines

Florian Moser

August 7, 2020

1 einleitung

algorithmen für / in verteilten systemen

1.1 verteilt

zustand & kontrolle an verschiedenen orten
verschiedene elemente kooperieren durch kommunikation
um bestimmten zweck zu erfüllen

1.2 verteiltes system

autonome programmierte einheiten (knoten, prozesse)
kommunizieren über nachrichten mit unbestimmter laufzeit
physisch distanzierte cores, machines, locations

logisch verteilt

abstraktion von physisch verteilt da distanz irrelevant
zustand verteilt
uhren nicht synchronisiert
keine instantane übermittlung

eigenschaften

keine globale sicht
keine gemeinsame zeit
unbestimmte nachrichtenlaufzeit

beispiel erde

gesamtzustand nicht erfassbar
globale zeit nicht erfassbar
kooperation durch kommunikation
viele parallele aktivitäten
ursache und wirkung getrennt (zeit/raum)

beispiele (mit zunehmender abstraktion)

rechnernetz mit rechnerknoten (compute-cluster, LAN, internet)
elemente im OS/middleware/programmiersprachen (prozesse, objekte)
protokollebene (aktionen, ereignisfolgen)

abstraktion zur protokollebene

ausreichend um probleme zu analysieren
visualisiere aktionen, ereignisfolgen
diskutiere konsistenz, korrektheit

herausforderungen algorithmik

heterogen, parallel, nichtdeterministisch, zustandsverteilt
schwieriges verständnis, komplexe programmierung/testing
braucht gute tools & methoden

1.3 verteilte berechnung

viele gleichartige, parallel instanzen
kooperation / synchronisation durch nachrichten
mehrere kontrollpunkte (programmzählstände)
mehrere mögliche ausführungen möglich
visualisierung durch zeitdiagramme (gummibandtransformation)

basisalgorithmen

bausteine für komplexe verfahren die garantien ermöglichen
bestimmen oft komplexität verfahren entscheidend mit
austauschbar nach voraussetzungen und gewünschten garantien
wie wellenalgorithmien, terminierungserkennung, ...

2 mathematische grundlagen

2.1 grundbausteine

logarithmus

es gelten die 1/exponentialfunktionengesetze
 $\log(a) + \log(b) = \log(a*b)$
 $\log(a^b) = b*\log(a)$
 $\log_a(b) = \log_c(b) / \log_c(a)$

harmonische reihe

$1 + 0.5 + 0.3333 + 0.25 + \dots \rightarrow \ln(n) + \gamma (= \sim 0.5)$
 $1 + 0.5 + 0.25 + \dots \rightarrow 2$

erwartungswert

$\sum \text{probability} * \text{result}$

2.2 relationen

bestimmte zuordnungen von elementen
zuteilung elementekombination mittels prädikat (true/false eigenschaft)
R untermenge A x B

karthesisches produkt

alle kombinationen der kombinierten mengen
 $A \times B \rightarrow$ alle tuples (a,b)

eigenschaften

transitiv ($a=b \ \&\& \ b=c \Rightarrow c=a$)
symmetrisch ($a=b \Rightarrow b=a$)
asymmetrisch ($a < b \Rightarrow \text{not } a < b$)
antisymmetrisch (asymmetrisch oder $a=b$)
totalität ($x < y$ or $x < y$)

relationen

halbordnung (reflexiv, transitiv, antisymmetrisch); ordnung
totalordnung (halbordnung + totalität)
äquivalenz (reflexiv, transitiv, symmetrisch); gleichheit

mengen

supremum (kleinste obere schranke; kleiner/gleich argumente)
infimum (grösste unterste schranke; grösser/gleich argumente)
verband/lattice (alle zwei elemente haben supremum/infimum)

verband

entsteht aus ordnungen
für jedes element existiert ein infimum/supremum
für endliche verbände visualisierung mit hasse diagramm
graph mit punkt pro element, kante wenn vergleichbar, y sortiert

transitive hülle

erweitert bestehende relation um "indirekte" relationen

2.3 graphen

eigenschaften

gerichtet (alle edges haben eine richtung)
azyklisch (keine zyklen)

arten

DAG (gerichtet & azyklisch)
bidirektional/unidirektional (kanten beide richtungen/nur eine)
vollständig (alle knoten mit allen anderen verbunden)
hamiltonisch (enthält pfad der alle knoten einmal enthält)

darstellung

adjazenzmatrix (x/y knoten, 1=edge vorhanden sonst 0)

2.4 topolgien

2d

linie (knoten nacheinander verbunden)
ringe (linie + letzter/erster knoten verbunden)
stern (zentrumsknoten mit allen verbunden)
baum (bei n knoten n-1 kanten & zusammenhängend)

hypercubes

würfel der dimension d
mit 2^d knoten und $d*2^{d-1}$ kanten
knoten mit bit pro dimension anschreiben
kante zwischen knoten die in einem bit unterschiedlich sind
maximale weglänge $d = \log n$; mittel $d/2$
routing mit sender XOR receiver

2.5 netzwerk

eigenschaften

anonym (knoten keine identitäten)

spannbaum

kantenauswahl die azyklisch alle knoten verbindet
bei n knoten genau n-1 kanten benötigt

2.6 primzahlen

satz von euklid

beweis für beliebig viele primzahlen
let m be multiplikation von allen bekannten primzahlen
 $m+1$ ist entweder primzahl (widerspruch)
oder $m+1$ enthält teiler aus bekannten primzahlen
dieser teiler muss sowohl m als auch $m+1$ teilen (widerspruch)

3 grundelemente

3.1 algorithmen

sequentiell

zur lösung eines problems
von zustand zu zustand
z.B. GCD

parallel

zur beschleunigung eines sequentiellen algorithmus
mehrere gleichartige prozesse (vielleicht mit master)
synchronisation über gemeinsamer speicher
z.B. divide & conquer

verteilt

gleichberechtigte prozesse synchronisieren über nachrichten
kooperation für gemeinsames ziel aber keine zentrale kontrolle

correctness

safety (something bad will never happen)
liveness (something good will eventually happen)
sonst oft triviale version der algorithmen möglich

3.2 probabilistische algorithmen

schwächen korrektheitsbegriff zu "partiell korrekt" ab

las vegas

partiell korrekt und terminierung mit w'keit 1
worst-case laufzeit unbeschränkt, dafür immer korrekt

monte carlo algorithmus

terminiert immer, partielle korrektheit w'keit $p < 1$
terminiert auch im worst case, dafür mögl. inkorrekt
anwendbar für p =klein & grosser effizienzgewinn
für suchprobleme & entscheidungsprobleme

3.3 modelle

vereinfachtes abbild der realität, potentiell realisierbar
vom urzustand durch gescheneisse (events) zu endzustand
interpretation realen gescheneisse für modellinput
interpretation output modell für schlussfolgerungen realität

nachrichtengesteuertes modell / transaktionsmodell (prozesse/nachrichten)

aktive prozesse versenden nachrichten oder werden passiv
passive prozesse werden durch empfang nachricht aktiv
common knowledge (alle gleicher zustand) unerreichbar

synchronmodell (nur prozesse)

aktive prozesse schalten andere aktiv oder werden passiv
lokale operationen mit laufzeit, nachrichten instantan
empfangereignis immer unmittelbar nach sendereignis
nachricht-scheduling relation zyklensfrei (send < receive)
nachrichtenpfeile senkrecht

atommodell/asynchron (nur nachrichten)

nur nachrichten unterwegs, die wiederum neue auslösen
lokale operationen instantan, nachrichten mit laufzeit

3.4 modelltransformation

gummibandtransformation

änderung diagramm einer verteilten berechnung
durch stauchen und dehnen der prozessachsen

alle resultierenden diagramme sind topologisch equivalent

nachrichtengesteuert → atom

virtuelle nachricht an sich selbst auf ende aktivitätsphase

nachrichtengesteuert → synchron

prozess bleibt aktiv bis explites/implizites ACK

atom → synchron

mit gummibandtransformation senkrechte nachrichtenpfeile machen
nur möglich bei kausaltreuen beobachtungen

3.5 zeitkomplexität

erlaubt aussage über ungefähre laufzeit algorithmus
legt zeit pro ausgetauschte nachricht fest
unter annahme atomarer lokaler berechnungen

einheitszeitkomplexität

wenn immer 1 zeiteinheit pro nachricht
zB broadcast laufzeit = weitester pfad vom initiator
aber unterschiedlich schnelle verbindungen nicht möglich

variable zeitkomplexität

wenn max 1 zeiteinheit pro nachricht; zB 1/n
zB bestimmte kanten nur 1/n laufzeit
erlaubt abbildung zusätzlicher szenarien
zB graph = $a \rightarrow b \rightarrow c$, $a \rightarrow c$; c crashed wenn von b empfangen
modellierung mit $a \rightarrow b$, $b \rightarrow c$ laufzeit 1/2, $a \rightarrow c$ laufzeit 1

3.6 deadlock

prozesse im system warten auf jeweils anderen
prozess muss beendet werden damit system fortschritt macht

phantom deadlocks

$t=1$ A blocks B
 $t=2$ B blocks C
 $t=3$ C blocks A
aber mögl. trotzdem nicht blockiert
ressource könnte im "gegenuhrzeigersinn" gewechselt haben
 $t=1$ at A, $t=2$ at B, $t=3$ at C

auflösung durch zufall

wie lösung blockade rechtsvortritt durch handzeichen
aber schwierig wenn alle algorithmen symmetrisch
bei kritischen algorithmen unerwünscht zwecks nachvollziehbarkeit

3.7 zustand

sammlung aller attribute zu bestimmtem zeitpunkt

lokaler zustand

kreuzprodukt variablenzustände
ereignisse transformieren lokale zustände

globaler zustand

kreuzprodukt lokaler zustandsräume
menge von nachrichten in transit
definiert zu bestimmtem zeitpunkt (wie endzustand)

definierung von zeitpunkt

triviale zustände wie start/endzeitpunkt
präfixberechnungen

3.8 prädikat

gelten im allgemeinen nur relativ zum beobachter
aber #zustände polynomiell, #beobachtungen exponentiell

qualifikation

prädikat aus sicht beobachter X erfüllt
prädikat von wenigstens einem beobachter erfüllt
prädikat von allen beobachtern wahrgenommen

beobachtungsvielfalt

verteilter algo kennt mehrere abläufe (nichtdeterminismus)
von gleicher instanz unterschiedliche beobachtungen möglich
(relativistischer effekt)
berechnung und beobachtung nicht zusammenfallend wie bei sequentiell

stabile prädikate

monotonie (für $c_1 < c_2$ gilt dass $\text{pred}(c_1) \Rightarrow \text{pred}(c_2)$)
beobachterinvariant; schliesslich sieht es jeder beobachter
wie kausale abhängigkeit, terminierung, deadlock

4 konsistente beobachtungen (causal-order)

möchte ursache vor (indirekter) wirkung beobachten

4.1 probleme

pumpe

druckmesser misst druckverlust und sendet an pumpe + beobachter
pumpe reagiert mit druckerhöhung und sendet an beobachter
aber mögl. druckerhöhung vor druckverlust bei beobachter
beobachter zieht schlussfolgerung dass pumpe grundlos erhöht hat

verteilte ampelsteuerung

2 ampeln; schalten jeweils autonom auf rot
darf auf grün schalten wenn die andere rot geworden ist
bei mehreren beobachtern uU andere kausalfolge beobachter

4.2 transitive halbordnung

ordnungsrelation auf einigen elementen
zur ermittlung von kausalität

halbordnung → lineare fortsetzung

konkrete mögliche ausprägung der halbordnung
mögliche empfindung für kausaltreuer beobachter

lineare fortsetzung → halbordnung

gegeben sei eine halbordnung
gegeben sei menge aller linearen fortsetzungen
schnitt aller elemente der menge ist halbordnung selbst

4.3 kausalrelation < ("happened before")

gerichtet & zyklensfrei
partielle ordnung (reflexiv, asymmetrisch, transitiv)
drei möglichkeiten, mindestens eine muss halten
(a) x & y auf selbem core & x vor y
(b) x ist sendereignis, y korrespondierendes empfangsereignis
(c) $\exists z$ such that $x < z \wedge z < y$
für alle $e < e'$ gilt, e' kennt / wurde beeinflusst von e

kausalkette

relation von e zu e' existiert wenn pfad existiert
pfad = nachrichtenpfeile + teilstücke prozessachsen

kausal unabhängig

$a \parallel b \Leftrightarrow \text{not}(a < b) \wedge \text{not}(b < a)$
nicht transitiv

4.4 beobachtungen

mehrere beobachtungen liefern uU anderes resultat
schlussfolgerung über stabile prädikate trotzdem möglich
auch wenn zustände zwischen beobachtungen unbekannt

probleme

unklar ob mehrere beobachtungen äquivalent sind
da zeitverzögerung beobachtung, verzerrung verhalten

idealer beobachter

empfangt alle beobachtungen in (kausaltreuer) reihenfolge
damit ursache/wirkung nicht vertauscht wird

kausaltreue beobachtungen

eine linearisierung der (partiellen) kausalordnung ($E, <$)
konsistent (ursachen stets vor wirkung; keine nachricht rückwärts)
beobachtung möglich, jedoch unentscheidbar ob tatsächlich so eingetreten
gummibandtransformation zur senkrechten linie möglich
beobachtete kausalrelation als objektive tatsache

unterschiedliche linearisierungen

da beliebige reihenfolge für kausal unabhängige ereignisse
da alle einverstanden mit kausalrelation keine bevorzugung möglich
schnitt aller linearisierungen ist wiederum kausalrelation
eigentliche beobachtung daher durch schnitt realisierbar

4.5 schnitte

schnitt

linksabgeschlossen bzgl. lokaler kausalrelation
falls $e' \in S \wedge e <_1 e' \Rightarrow e \in S$
keine aussage über ereignisse anderer knoten
mögl. inkonsistent; nachrichten aus der "zukunft"
baustein für terminierungserkennung, schnappschuss, ...

konsistenter schnitt

linksabgeschlossen bzgl. kausalrelation

falls $e' \in S \wedge e < e' \Rightarrow e \in S$
ist immer ein schnitt, weil $<_1 \text{part}_of <$
gummibandtransformation möglich
konsistente schnitte \Leftrightarrow kausaltreue beobachtung

kausale vergangenheit

relation r die alle e' die kleiner e sind enthält
konsistenter schnitt
 $e' < e \Leftrightarrow e' \in r(e)$
 $e \parallel e' \Rightarrow$ nicht in r des jeweils anderen
bezeichnet den "kegel" von e

4.6 synchrone kausalitätsrelation $<<$

ist eine halbordnung
für synchronmodell

definition

für korrespondierende vorgänge s und r gilt
(1) falls a vor b auf selbem prozess dann $a << b$
(2) $x << s \Leftrightarrow x << r$ (common past)
(3) $s << x \Leftrightarrow r << x$ (common future)
(4) transitiver abschluss

interpretation

s/r mögl. nicht in relation
 $<<$ gibt aber eine lineare erweiterung vor

lineare erweiterung

linearisierung der vorgänge
sodass für alle korrespondierende vorgänge s und r gilt
dass s direkter vorgänger r ist (da ja blockierend versand wird)
bei deadlock $s_1 \rightarrow r_1, s_2 \rightarrow r_2$ mit $r_1 > s_2$ und $r_2 > s_1$ unmöglich
denn alle vier linearisierungen widersprechen sich

5 verteilte berechnung

5.1 definitionsversuche

ordnungsrelation ($E, <$)

aber ereignisse auf unterschiedlichen cores nicht unterscheidbar
partitionierung von E auf cores zu E_i nötig

ordnungsrelation ($E_1, \dots, E_n, <$)

E_i paarweise disjunkt; jeweils ereignisse auf einem core
 $<$ als irreflexible halbordnung auf E_1, \dots, E_n
 $<$ als lineare ordnung in E_i
jedoch FIFO / nicht FIFO noch nicht unterscheidbar
daher unterscheidung sender/receiver nötig

5.2 definition

n -fache verteilte berechnung mit asynchroner kommunikation
(1) partitionierung E in E_i je nach core
(2) partitionierung E in sende/empfangsereignis $SxR = T$
 S & R leere schnittmenge (kein kombiniertes senden/empfangen)
für jedes r gibts genau ein s dass $(s,r) \in T$
für jedes s gibts höchstens ein r dass $(s,r) \in T$
("höchstens" modelliert message loss; erlaubt interne ereignisse)
(3) $(s,r) \in T \Rightarrow s < r$
(4) $<$ ist lineare ordnung in jedem E_i
(5) $<$ ist irreflexible halbordnung auf E (no time travel)
(6) $<$ kleinste relation (keine weiteren ereignisse in relation)

5.3 präfixberechnung

gegeben berechnung $B = (E_1, \dots, E_n, L, <)$

für B'

E' teilmenge E , linksabgeschlossen
 L' & $<'$ sind einschränkungen von L & $<$
 E' ist linksabgeschlossen
wenn $x \in E'$, dann $\in y < x \Rightarrow x \in E'$
somit bleiben kausalrelationen erhalten (nur "echte" prefixe)

eindeutig bestimmt durch

(a) alle inkludierten ereignisse E' oder
(b) jeweils letzte lokale ereignisse (weil linksabgeschlossen)

präfixrelation

gerichtet, zyklensfrei, transitiv \Rightarrow eine halbordnung
mögl. mehrere möglichkeiten für direkten vorgänger
daher ablauf der berechnung nicht zwingend eindeutig definiert
bei sequentiell werden zwingend alle präfixe durchlaufen

präfix-verband

für jeden prozessor kommt dimension hinzu
jeweils verbindung von prefixen zueinander
somit hat zwischenstand idR mehrere mögliche vorgänger
berechnung bewegt sich undefiniert von start zu endzustand

als gitter

x-achse geordnete ereignisse p_1 , y-achse von p_2
schnittpunkte als mögliche beobachtungen / abläufe
wirklicher ablauf von links unten nach rechts oben
beobachtbare zustände folgt genereller richtung
einige schnittpunkte an rändern unmöglich wegen kausalität

als verband

konsistente zustände bilden verband (gemeinsamer sup/inf)
aus beobachtungen berechnung ungefähr nachvollziehbar
mit "flaschenhälsen" zustände in äquivalenzklassen einteilbar
dabei unklar welcher der zustände tatsächlich erfüllt war
nur dass einer davon sicher durchschritten wurde
äquivalenzklassen erlauben aussagen ("barrier synchronisation")

6 verteilte approximation

kontinuierliche publizierung besseres standes

6.1 ablauf

approximation verbessern bei empfang nachricht
mittels assoziativen operatoren (min, max, union, intersect, +, -, ...)
informiere nachbarn initial & bei besserer approximation
laufende verbesserung der approximation; monoton bei monotonen operatoren
stagnation/terminierung bei optimum

6.2 beispiele

bellman-ford

kürzester weg zu anderen knoten finden
knoten hat tabelle mit ziel, next hop, cost
sendet initial & bei änderung an nachbarn (ausser sender)
bei empfang tabelle anpassen wenn kürzerer pfad gefunden

grösster gemeinsamer teiler (ggt)

knoten sendet eigenen teiler x zu nachbarn
wenn empfangenes $y \geq x$ dann ignorieren
sonst sende $x = \text{mod}(x-1, y) + 1$ allen nachbarn
terminierung wenn alle gleicher wert haben

paralleles zahlenrätsel

prozesse mit $a*x + b*y = c*z$ für a, b, c beliebig
constraint-propagation (eigene einschränkungen verbreiten)
backtrack (um lösung zu erarbeiteten hypothesen zu finden)
mögl. nicht monoton abnehmend, mehrere lösungen

7 broadcast

alle knoten über etwas informieren dann terminieren
mind n-1 nachrichten benötigt

7.1 nach topologie

für ring wenn token zurück beim initiator
für spannbau wenn n-1 nachrichten versandt
für beliebig algorithmus wie flooding, echo

7.2 anwendungen

zur verbreitung information
zum kennenlernen nachbarschaftsidentitäten

7.3 beliebige topologien

flooding

initiator (beliebig) sendet basismnachricht an alle
bei erstem empfang basismnachricht an alle weiterleiten
 $2e - n$ (aktivierungskanten) + 1 (initiator) nachrichten
terminierung indem beobachter auf ACK aller wartet
oder ACK nur pro knoten (aber mögl dann noch nachrichten unterwegs)
oder überlagerung kontrollalgorithmus

echo (chang)

initiator sendet basismnachricht an alle, wartet auf quittung
bei empfang basismnachricht diese an alle nachbarn weiterleiten

quittung zurücksenden wenn für eigene basismnachrichten erhalten
quittung für alle basismnachrichten ausser ersten sofort versenden
terminierung wenn initiator alle quittungen empfangen hat
maximal $4e$ nachrichten (cycles treffen sich)

echo (PIF von segal)

alle knoten starten weiss
flooding von "explore", gestartet von initiator ("rot werden")
"echo" über empfangskante zurücksenden ("grün werden")
wenn rot & alle kanten "echo"/"explore" erhalten haben
einziger initiator oder unterschiedliche farben notwendig
 $2e$ nachrichten (explorer+echo oder explorer+explorer)
weisse phase disjunkt roter (rote hinwelle, grüne rückwelle)
empfangskanten ("grüne kanten") bilden schnellster spannbau

verbesserung explorer-reduce

braucht identitäten der nachbarn, $O(n)$ nachrichtsgrösse
mit explorer tabu-menge (nachbarn, vorgänger) mitsenden
an diese werden von empfänger nichts mehr versandt
spart explore an vorgänger, nachbarn
keine ersparnis bei parallelen wegen abkürzungen

zeitkomplexität PIF auf vollständigem graph

mit einheitszeitkomplexität in $t = 3$
weil $t=1$ alle rot, $t=2$ alle grün, $t=3$ initiator fertig
mit variabler zeitkomplexität $t = n$ in worst case
sei $t=1/n$ für explorer ring aussen, sonst $t=1$
dadurch wird auf ring bau aufgebaut, fertig bei $t=1$
danach wandern echos auf bau zurück, braucht $n*1$ einheit

7.4 hypecubes

dimension broadcast

jede iteration in entsprechende dimension senden
d iterationen mit $2^d - 1$ nachrichten (optimal)
 $000 \rightarrow 001$
 $000 \rightarrow 010, 001 \rightarrow 011$
 $000 \rightarrow 100, 001 \rightarrow 101, 010 \rightarrow 110, 011 \rightarrow 111$
aber slow start, heavy use am ende

parallel broadcast

jeweils an alle "höheren" nachbarn (0-prefix)
somit zB für 3er 0-prefix an 3 nachbarn
d iterationen mit $2^d - 1$ nachrichten (optimal)
 $000 \rightarrow 001, 000 \rightarrow 010, 000 \rightarrow 100$
 $001 \rightarrow 011, 001 \rightarrow 101, 010 \rightarrow 110$
 $011 \rightarrow 111$
frühe information, wenig parallele nachrichten

d-message broadcast

broadcast mit d nachrichten gleichzeitig
wie dimension broadcast & jedes packet wählt andere dimension

8 verteilte terminierung

feststellen ob algorithmus terminiert
für transaktionsmodell falls nicht anders definiert

8.1 beispiel feuerdörfer

feuer in dörfern, das ab und zu erlischt
feuerbote kann feuer in anderes dorf transportieren
anderer bote besucht dörfer und sucht nach feuer
bote beobachtet immer $\#sent = \#received$ feuerboten & nie feuer
kann dennoch nicht davon ausgehen dass nirgendwo feuer brennt
da er kausalgrenze möglicherweise überschritten hat
wie wenn bote/feuerbote gleichzeitig umgekehrt unterwegs

8.2 terminierung nach modell

transaktionsmodell wenn knoten passiv, keine nachrichten
atommodell wenn keine nachricht unterwegs
synchronmodell wenn alle prozesse passiv

anforderungen modelltransformation

wenn transformation beendet, dann auch original (safety)
wenn original, dann auch transformation (liveness)

8.3 "stop the world"

alles einfrieren
terminiert wenn $\#sent/\#received$ gleich & alle prozesse passiv
sonst alles wieder auftauen

8.4 fehlerhafte algorithmen

beobachten uU schiefes bild

zählen empfang/versand

knoten a,b,c; $a \rightarrow b$, $b \rightarrow c$; besucht knoten zu spezifischen zeitpunkten
b vor empfang von $a \rightarrow b$, a nach versand $a \rightarrow b$, c nach empfang $b \rightarrow c$
beobachter stellt terminierung fest, da #sent/#received gleich
fails da nachricht aus der zukunft gesehen wird
weil schnitt kausalgrenzen überqueren kann

beobachter über empfang/versand informieren

knoten a,b,c; $a \rightarrow b$, $b \rightarrow c$, $b \rightarrow a$; nachricht an beobachter bei aktivität
kontrollnachricht wird empfangen von a dann c
beobachter stellt terminierung fest, da #sent/#received
fails da kontrollnachricht von b zu spät eintrifft
weil beobachtungsnachrichten sich überholen können

8.5 nachrichten eindeutig benennen

nachricht bekommt uid, knoten merken sich alle sent/received
terminierung wenn auf schnitt alle versandten empfangen worden sind
funktioniert da dann keine aktivität nach schnitt stattfinden kann

eindeutige namen

mit knoten- oder kanalidentifizierung
hierarchisch (knoten.kanal.#sent)

8.6 zählen pro unidirektionalem FIFO kanal

für kanal = (source, target) (ohne umkehrung)
terminierung wenn #sent = #received

konsistenzbeweis durch widerspruch

für A kanal und A.x nachricht x auf diesem Kanal
annahme ereignis innerhalb schnitt (A.x)
löst frühestes ereignis nach schnitt aus (B.y)
damit #sent = #received muss ein ereignis aus der zukunft (A.a)
ein ereignis vor schnitt auslösen (B.b)
es gilt widersprüchlich dass (A.a) < (B.b) < (B.y)

8.7 doppelzählverfahren

zwei beobachtungswellen W_1 W_2 die #sent und #received zählen
terminiert wenn $\#sent_1 = \#received_1 = \#sent_2 = \#received_2$
somit war während $W_1 < t < W_2$ keine nachricht unterwegs
falls $W_1 = W_2$ kann W_2 als neue W_1 benutzt werden

beweis

für $E \sum(\text{empfängerzähler})$, $S \sum(\text{senderzähler})$
für #received erste welle = #sent zweite welle
 $\Rightarrow E(\text{nach erster welle}) \geq S(\text{vor zweiter welle})$
 $\Rightarrow E(\text{nach erster welle}) \geq S(\text{nach erster welle})$ da früher
 $\Rightarrow E(\text{nach erster welle}) = S(\text{nach erster welle})$ da $E(t) \leq S(t)$

eigenschaften

reentrant; prozessstate wird durch algorithmus nicht geändert
kontrollrunden unabhängig von senden/empfangen basisnachrichten

initiator

jeder prozess kann eigene runden starten
optimalerweise knoten der nach empfang nachricht passiv wird

kontrolltopologien

ring / hamiltonischer zyklus ("logischer" ring; zwei durchläufe)
spannbaum (jeweils an blättern reflektieren; zwei durchläufe)
echo-algorithmus (einzige ausführung genügt da bereits zwei wellen)

8.8 zeitzoneverfahren

entlang des schnittes zählen + erkennung nachrichten aus zukunft
damit inkonsistente schnitte erkennen

algorithmus

prozess setzt flag bei empfang nachricht aus höherer zeitzone
welle erhöht zeitzone des prozesses, setzt flag zurück
terminierung wenn ganze welle kein flag & #sent=#received

schwarz-weiss zeitzone

nur schwarz/weiss unterscheiden (statt n zeitzone)
jede nachricht aus zukunft wird erkannt (safety)
jedoch mögl. false positives aus vergangenheit (no liveness)
u.U. einzige zusätzliche runde vonnöten

eigenschaften

nicht reentrant (da basisnachrichten verändert)
braucht eingriff in basisnachrichten (zeitzone mitsenden)

vermeiden inkonsistenter schnitt

wenn bekannt das nachricht n aus zukunft
wird schnittlinienoperation wie vor empfang n simuliert
somit kann schnittlinie alte werte mitgegeben werden
zusätzliche runde kann damit u.U. wieder eingespart werden

8.9 DFG terminierung (synchronmodell)

mit welle durch passive prozesse gehen
dabei aktivierung anderer prozesse entdecken
braucht eindeutiger initiator, oder färbung pro initiator

algorithmus

knoten mit eindeutiger id, farbe = weiss
wenn knoten nachricht an höhere id sendet, farbe = schwarz
token läuft von gross zu klein (höchster initiiert)
token anfänglich weiss, wird schwarz wenn bei schwarzem knoten
knoten wird nach besuch token weiss
terminiert wenn nach durchlauf token immer noch weiss

bei asynch kommunikation

nicht möglich (da schwarz-färben sich überlagern könnte)
modeltransformation nach synchron mich ACKs

beispiel

W_1 setzt alles auf weiss, trennt keine aktive prozesse durch
bei empfang basisnachricht jeweils schwarz färben
 W_2 überprüft dass kein flag gesetzt wurde
terminierung wenn W_1 und W_2 erfolgreich
 W_1/W_2 kombinieren (flag prüfen & zurücksetzen) möglich

8.10 parallelisierung

arbeitseinheit entsteht bei iterativer, rekursiver berechnung
neue arbeitseinheiten werden an andere prozesse ausgelagert
sind alle arbeitseinheiten fertig ist berechnung abgeschlossen
terminierung wenn alle knoten last (#arbeitspakete) = 0

beispiel integration

prozesse integrieren bestimmtes intervall
intervall halbieren um arbeitspakete für andere zu generieren
resultat & intervallgrösse an akkumulatorprozess senden
terminierung wenn akkumulatorprozess #intervallgrösse = 1

ticketsystem

fixe anzahl tickets vorhanden
teilnehmer bei eintritt system muss ticket beziehen
teilnehmer bei austritt system muss ticket abgeben
wenn tickets am ausgang vollständig, dann bereich leer
wenn zu wenig tickets, halbierte tickets ausgeben

8.11 kreditmethode

methode zur terminierungserkennung bei parallelisierung

vorgehen

urprozess startet berechnung
prozesse / nachrichten haben jeweils anteil > 0
summe aller anteile jeweils 1
terminiert wenn urprozess wieder kredit 1

korrektheit

bedingungen müssen jederzeit erfüllt bleiben (safety)
am schluss muss kredit wieder beim urprozess sein (liveness)

realisierungsoption

bei passiv werden wird kredit dem urprozess zugesandt
bei empfang nachricht wird kredit empfänger zugerechnet
bei versand nachricht wird hälfte kredit sender mitgegeben

implementierung

2-er exponent des nenners merken (halbierung daher $a = a + 1$)
rekombination durch addition "von vorne"
wie $0.01 + 0.01 = 0.1$
aber bitlänge kann sehr lange werden

implementierung mit komplementen

merkt sich an welcher position noch bits fehlen
start mit $D = \{0\}$
(bei empfang von ID)
while $ID \text{ not_element_of } D$ then add ID to D; ID-
remove ID from D
if D empty then terminate
 $1/4 + 1/2 + 1/8 + 1/8 \Rightarrow D = \{2,1\}$, $D = \{2\}$, $D = \{3\}$, $D = \{0\}$

komplexität

$O(m+n)$ nachrichten (optimal)
könnte kreditanteile durch wellenalgorithmus einsammeln

erweiterung auf nachlaufmethode

kontrollnachrichten zur rückforderung von krediten
auf benutzten FIFO kanälen mit basisnachrichten hinterher senden
"geeignetes" aufspalten bei prozessen
rückkehr zur urprozess wenn kanäle inaktiv werden

analogie nachlaufmethode zu echo

rückgabe wenn empfangen passiv wird & alle ACKs empfangen
kreditwerte werden nicht mehr gebraucht (da immer received=sent)
"explorer" als arbeitspakete, "echo" zur terminierung

8.12 arora algorithmus (falsch!)

terminierungstoken wandert im kreis
nodes überwachen jeweils zustand nachbarn (aktiv/passiv)
terminieren wenn alle nachbarn passiv

gegenbeispiel

terminierungstoken verlässt p_j , wird über p_k p_i erreichen
 p_i sendet zur gleichen zeit auftrag an p_j
 p_j aktiviert andere & terminiert
 p_i terminiert da alle nachbarn terminiert sind
 \Rightarrow weil nur nachbarstate getracked wird

9 verteilter wechselseitiger ausschluss

geteilte ressource mit beschränkter gleichzeitiger verwendung
"alle wollen, einer darf"

9.1 anforderungen

safety (nie mehr als eine instanz zugelassen)
liveness (mindestens eine instanz zugelassen)
fairness (jeder request wird schliesslich erfüllt)

9.2 qualitätskriterien

nachrichtenkomplexität (#sent, #received)
syntaktische symmetrie (gleiches verhalten)
semantische symmetrie (gleiche last)
fehlertoleranz (nachrichtenverlust, deadlock, mitigationsaufwand)
fairness (abarbeitung relativ zur globalen reihenfolge)
zeitbedarf (scheduling overhead; auch relativ zur last)
effizienz implementierung (topologievoraussetzungen)

9.3 triviale lösungen

monitor (simpel)
alle fragen (dezentral, symmetrisch aber viele nachrichten)
auswahl fragen (weniger nachrichten, aber unklare umsetzung)

9.4 klassifizierungen

token basiert

safety trivial, aber fairness nicht
unterschiedliche anforderungsmechnismen, topologien
braucht election für tokenverlust, a-priory unbekannten ressourcen

request basiert

request set wird gefragt um erlaubnis
aber safety, deadlock nicht trivial

zentraler manager

eine instanz vergibt zutritt
wie request/token basiert zusammen
simpel aber schlecht skaliertbar

hierarchische / hybride verfahren

bei grossen systemen mehrere ebenen/clustern
diese benutzen mögl. jeweils eigenes verfahren

9.5 maekawas

anordnung der prozesse in \sqrt{n} gitter
jeder prozess fragt spalte & zeile um erlaubnis
menge von angefragten prozessen $2\sqrt{n} = |R_i|$
request, grant & release nachrichten (somit $3 |R_i|$)

deadlocks

sind möglich, insb. wenn prozesse nur ein schnittpunkt haben
dies wenn "im kreis" blockiert wird

verfahren zur verhinderung umsetzbar mit nur konstantem faktor

\sqrt{n} mit projektiver geometrie

\sqrt{n} ist optimal für symmetrische verfahren
projektive geometrie ändert geometrie
+ punkt im unendlichen für parallele geraden
+ gerade die durch alle "unendlichen" punkte geht
projektive ebene der ordnung k verlangt dass
(1) k geraden genau k punkte enthalten
(2) ebene hat $k(k-1)+1$ punkte und gleich viele geraden
für $k=2$ 3 geraden (zwei parallel); mit jeweils 2 schnittpunkte
somit 3 quoren, die jeweils 2 prozesse anfragen
für $k=4$ ergibt es 13 quoren, jeweils 4 prozesse anfragen
aber quoren unmöglich/schwierig zu finden für beliebige k

$\sqrt{2} \cdot \sqrt{n}$ menge

prozesse dreiecksförmig anordnen (90° links unten)
durchnummerierung zeilenweise von oben
zeile + erste getrennte spalte als quorum (schema 1)
quorum für $N=10$, $p=3$ ist 2,3 (zeile) + 6,9 (spalte)
spalte + erste getrennte zeile als quorum (schema 2)
quorum für $N=10$, $p=3$ ist 3,5,8 (spalte) + 1 (zeile)
schema 1 belastet hohe zahlen mehr, schema 2 tiefe
für symmetrische nutzung der nodes schemas alternieren

9.6 token ring

token wird weitergegeben sobald ressource frei
valides und simples verfahren ohne deadlocks
aber ständiges kreisen, lange wartezeiten, möglicher tokenverlust

broadcast basiertes request

broadcast an alle das token benötigt wird
token wandert so nur bei bedarf

ricard/agrawala

token registriert $visited_{time}$ pro prozess
token-request broadcast; im token mit $last_request_{time}$ vermerken
wird token frei an prozess senden der am längsten wartet
skip wenn $last_request_{time} < visited_{time}$ auf token

spannbaum basiertes request

statt broadcast auf spannbaum nach token suchen
max $n-1$ nachrichten bis token gefunden

9.7 lift algorithmus

statt broadcast zu token gerichteter spannbaum verwenden
kanten zeigen immer in richtung tokenbesitzer
nutzt token eine kante ändert es deren richtung
tokenanforderung über beliebige ausgehende kante(n)

9.7.1 verbesserung anfragenverarbeitung

prozess sendet/leitet nur einzige anfrage weiter
merkt sich bis token erhalten wer alles angefragt hat
dann an "faire" richtung weiterleiten
bei mehreren richtungen, token direkt erneut anfordern

9.7.2 komplexität nach topologie

grad-k knoten

$\log_k(n)$ als längster weg
 $\log_k(n)$ auch als mittlere weglänge
bei grosser last weniger (da immer "treffer")
innere knoten werden stärker belastet

stern

kürzester weg, aber single point of faillure
uU. konstruktion zusätzlicher verbindungen nötig (vs spannbaum)

lineare kette

hat mittlere weglänge $n/3$

9.7.3 verallgemeinerung allgemeiner graph

gerichtet, azyklisch, zusammenhängend

konstruktion initialem graph

flussrichtung echo algorithmus gibt baumkantenorientierung vor
orientierung übriger kanten anhand (entfernung root, ordnung knoten)

invariante zyklensfreiheit

beweis durch widerspruch
neue kanten können nur durch invertierte kanten entstehen
diese zeigen aber immer auf token (durch invert beim traversieren)

request holt token immer ein

request wandert dem pfad entlang
immer nur ein ausgang pro knoten
(knoten kann daher keinen kreis bilden)
request kommt knoten auf weg immer näher
hin-und-zurück mit token nur auf pfad möglich

aktion bei ankunft knoten

nachbarn informieren kanten zu ändern (für kürzere wege)
richtung ankommender kante ändern (für wenig aktionen)
all anderen ausgehenden kanten löschen (für weniger wege)

10 election problem / symmetriebrechung

unter gleichen prozessen leader auswählen
symmetrie ohne deadlocks brechen

10.1 modell

n identische prozesse mit unique identity
fairness keine bedingung (!= wechselseitiger ausschluss)
 $k \leq n$ initiaten; rest bleibt passiv
 $k=1$ sollte minimale nachrichtenzahl implizieren
provozieren in der praxis mit zufälligen timeouts (wie ethernet)
 $k=n$ als theoretisch interessante worst-case complexity
 $k>1$ mit grundidee dass grösster prozesses die anderen passiviert
sparsamkeit bezgl. nachrichtenzahl

10.2 anwendungen

auswahl prozess um verfahren zu starten (wie generierung token)
symmetrisierung anderer verfahren
wie root auswählen um stammbaumerstellung zu initiieren
identitäten vergeben (leader sendet 1.i an knoten i)

10.3 algorithmen

knoten haben identität; grösster knoten soll leader sein
jeder knoten darf beliebig initiieren und muss endresultat kennen

mit verteilter approximation

einfach immer über höchste bekannte zahl infomieren
terminierungserkennung z.B. mit echo algorithmus
echos schlucken wenn bekannte zahl höher wie initiator
dann aber terminierung unklar

naive

message driven, atomar
jeder prozess identität p, $M = 0$
iff $M=0$ then $M=p$, send M to all (spontan)
(on message j received)
iff $j > M$ then $M = j$, send M to all
(on termination)
iff $M=p$ then leader = true

10.4 bully-algorithmus (unidirektionaler ring)

naive

jeder prozess sendet eigene identität in den ring
nachricht wird markiert wenn grössere identität gefunden
grösster prozess hat unmarkierte nachricht
 n^2 nachrichten

verbesserung message extinction (chang roberts)

nachricht löschen wenn höchste gesehene nummer grösser
wird eigene nachricht wieder empfangen dann leader
mittlere weglänge für n knoten / k starters $n H(k)$
worst case $O(n^2)$ bei n starters
"average case" optimal bei n starters
bei grösseren ringen fast nie mehr als $n H(n)$ nachrichten

verbesserung probabilistisch bi-directional (lavault)

senderichtung ziehen (50% ob rechts oder links)
in 50% der fällen kommt der eliminator entgegen
dadurch werden 25% der nachrichten gespart ($0.75 n \ln(n)$)
beweis ergibt sogar nur $0.71 n \ln(n)$
weil "high order eliminators" (hohe zahl eliminiert früh andere hohe)

10.5 nachrichtenkomplexität chang roberts

n zufallszahlen, suche #left-to-right maxima
urnenmodell ohne zurücklegen (keine korrelationen)

> i-te stelle erreichen

1/i weil muss grösser wie i vorangehen zahlen sein

somit mittlere anzahl $1 + 1/2 + 1/3 + \dots = H(n)$
somit zeitintervall bis nächster rekord $e = 2.7$ mal grösser als vorheriges

= i-ter stelle rekord

$1/i$ (i-ter stelle rekord) - $1/(i+1)$ (i+1 stelle rekord)
 $1/n$ (durchlauf) + $\sum (1/i(i+1)) = 1$
addition valid weil zufallsvariablen paarweise unabhängig

mittlere weglänge

weglänge mit wahrscheinlichkeit gewichten
 $0 + 1*1/2 + 2*1/3 + \dots = H(n) = \ln(n)$

für n nachrichten, k initiatoren

average case $n * H(k) = n * \ln(k)$
worst case wenn ring in umgekehrter reihenfolge
dann legt jede nachricht maximale strecke zurück
grösser braucht n, grösster-1 braucht n-1, ...
 $nk - k(k-1)/2$ für $k = \#stater$

10.6 hirschberg-sinclair (bidirektionaler ring)

prozesse senden id links/rechts zu 2^i knoten entfernung in runde i
grössere identität auf weg sendet veto zurück; dann passiv werden
sonst sendet ziel ein echo zurück; dann nächste runde starten
passive knoten leiten nur nachrichten weiter
terminieren wenn zwei mal eigene identität empfangen wird

nachrichtenkomplexität

kann nur kette von 2^i starten, wenn 2^{i-1} überlebt
innerhalb 2^{i-1} kann höchstens ein prozess starten
somit $\max n/(1+2^{i-1})$ starter bei runde i
jeweils $\max 2$ (hin/her) * 2 (links/rechts) * 2^i (segment) nachrichten
 $\Rightarrow \max 8 n \log(n) = 11.53 n \ln(n)$ nachrichten

zeitkomplexität

jeweils 2^i nachrichten warten pro runde
und weil $2 + 4 + \dots + 2^{i-1} < 2^i$
 $\max 1 + \log(n)$ phasen
 $\Rightarrow 2^{(1+\log(n))} = 4n$

10.7 peterson's election algorithm (bidirektionaler ring)

prozesse starten aktiv und senden identität links/rechts
zwei nachrichten schlucken; wenn beide kleiner wiederholen sonst passiv werden
passive knoten leiten nur nachrichten weiter
terminieren wenn eigene identität eintrifft

beispiel sortierte anordnung

erste phase elimiert alle ausser grösster identität
da alle anderen eine jeweils grössere identität empfangen
bei zweiter phase folgt terminierung
 $\Rightarrow 4n$ nachrichten

worst case komplexität

pro phase laufen 2 nachrichten über jede kante
phase halbiert anzahl aktiver knoten
keine drittelung; dabei würden knoten doppelt gezählt
 $\Rightarrow 2 \log(n)$ nachrichten

mittlere komplexität

im mittel wird mit wahrscheinlichkeit $1/3$ überlebt
somit $\log_3(n)$ phasen
 $\Rightarrow 2 n \log_3(n) = 1.82 n \ln n$ nachrichten

variante unidirektionaler ring

effiziente simulation des peterson algorithmus möglich
alte identität speichern, danach in ringrichtung verschieben
simuliert nachrichtenversand entgegen ringrichtung
nachrichtenversand in ringrichtung trivial

variante richtung abwechseln

phasen im und gegen uhrzeigersinn wechseln sich ab
nun nur noch n nachrichten pro phase
aber anzahl phasen $r \leq \log_{\phi}(n) + O(1)$ für $\phi = (1+\sqrt{5})/2$
abschätzung indem zwei phasen berücksichtigt werden
spart insgesamt nachrichten

10.8 election (bäume)

phasen

explosion (flooding zu blättern); reflektiert an blättern
von mehreren orten gleichzeitig möglich; ergibt "zentrumskante"
kontraktion (maximum aus identitäten zurück zum zentrum)
kann sich zeitlich mit explosionsphase überschneiden
information (optional; zentrum informiert andere)

flooding je nach anforderung algorithmus

komplexität

für k initiatoren

explosion in $n-2+k$ (k-1 explosionskanten)

kontraktion n (alle kanten jeweils eine; zentrums-kante 2)

informationsphase n-2 (zentrums-kante kann ausgelassen werden)

$\Rightarrow 3n+k-4$; für $k=1$ $3(n-1)$

vergleich zu ringen

auf bäumen election effizienter

ring in baum umwandeln indem kante entfernt wird

jedoch dafür wiederum election vonnöten

10.9 election (bidirektionale zusammenhängende graphen)

zwei entscheidungen

wie flooden (z.B. echo-algorithmus, ring)

wie nachrichten sparen (z.B. message extinction)

echo algorithmus

initiatoren starten algorithmus

explorer/echo tragen identität des initiators

schwächere nachrichten werden verschluckt

\Rightarrow stärkste welle erreicht alle

10.10 election vs spannbaumkonstruktion

election

mindestens e nachrichten nötig (für $e = \#kanten$)

wenn kein globales wissen, deterministisch

widerspruchsbeweise für $\#kanten < e$

(1) unbesuchte kante von stern verbirgt könig

(2) unbesuchte kante verbindet zwei partitionen

verteilte spannbaumkonstruktion

kann dezentral fragmente bilden und dann vereinen

möglich in $5e + 5n \log(n)$

schwierig ist zyklenverhinderung und sparsamkeit nachrichten

oder election & dann gewinner spannbaum festlegen lassen

bei echo-election spannbaum des gewinners bereits bekannt

komplexitätsvergleich

spannbaum \leq election + $2e$ (weil echo)

election \leq spannbaum + $O(n)$ (weil election auf bäumen)

wobei zusatz (+) jeweils vernachlässigbar

somit probleme vergleichbar schwierig

somit spannbaumkonstruktion unter $\omega(e)$ unmöglich

10.11 election für anonyme netze

wenn symmetrischer anfangszustand & verhalten

brechung der symmetrie nicht möglich

\Rightarrow probabilistische algorithmen verwenden

mögl. inkorrekt (monte carlo) oder keine terminierung (las vegas)

10.12 las vegas election für anonyme ringe der größe 2

random wird generiert und mit nachbarn verglichen

wenn unterschiedlich terminieren, sonst wiederholen

las vegas da korrekt aber keine terminierung bei gleichem random

mittlere komplexität

abhängigkeit next state nur vom vorherigen (markov chain)

beide wählen 0/1, dann vergleich

somit $p=0.5$ dass beide unterschiedlich (success)

und $p=0.5$ dass beide gleich und darum wiederholt werden muss

abschätzen somit mit $W = 0.5 + 0.5(1 + W)$

ergibt $W=2$; somit 4 nachrichten (da hin und zurück)

10.13 las vegas election für anonyme ringe der größe n

random id durchlaufen lassen; message extinction wenn nicht max

bei alleinigem durchlauf terminieren, sonst mit gewinnern wiederholen

las-vegas algorithmus da korrekt aber keine terminierung bei gleichen ids

wenn n unbekannt mit echo algorithmus herausfinden

algorithmus

eigene identität mit random(1, ...n) wählen

nachrichten mit identität i und hop-counter hc

bei nachricht mit $i < i_{own}$ nicht weiterleiten (message extinction)

bei nachricht mit $i > i_{own}$ weiterleiten mit erhöhtem hc

bei nachricht mit $i == i_{own}$ && $hc < n$ flag setzen

bei nachricht mit $i == i_{own}$ && $hc = n$ flag terminieren

falls kein flag bei terminierung gesetzt, dann gewonnen

sonst neue runde unter gewinnern starten

komplexität

$E[\text{rundenanzahl}] < e(n/(n-1))$

11 garbage collection

wurzel (wie static variable) hat pointers zu anderen objekten

garbage ist von keiner wurzel mehr erreichbar

anwendung bei dynamischem speicher / objektorientiert

stabiles prädikat (einmal garbage, immer garbage)

11.1 anforderungen

safety (eingesammeltes objekt ist garbage)

liveness (garbage wird schliesslich eingesammeln)

11.2 modell

collector sammelt ein um speicher wiederzuverwenden

mutator führt program & referenzoperationen gleichzeitig aus

referenzoperationen sind "new", "copy", "delete"

reduziertes model behandelt new/copy gleich

11.3 mark & sweep

von wurzeln aus alle erreichbaren objekte markieren

alle unmarkierten objekte einsammeln

mit "stop the world"

mutator anhalten

mark & sweep durchführen

mutator wieder starten

aber schlechte lösung für real-time/interaktive anwendungen

concurrent/parallel/on-the-fly

collector könnte referenz zu knoten nie sehen

wenn source während überprüfung entsprechend wechselt

daher muss mutator collector helfen

11.4 verteilte garbage collection (globale GC)

prozessorüberschreitende referenzen (remote ref)

entstehen durch entsprechende nachrichten (remote copy)

lokale GC unter annahme dass alle remote ref valid sind

globale GC bei der remote ref auch überprüft werden

mehrere mutators/collectors

mutator operationen

p sends copy(r) to q (send copy)

q receives copy(r) (receive copy)

q removes r reference (delete)

analogie verteilte terminierung

"objekt is garbage", "process is passiv" als stabile prädikate

kontrollalgorithmus entdeckt zustand parallel zur anwendung

"send copy", "send message" als kommunikation

gleiches problem der behind the back activation

11.5 reference counting

zählen wie oft auf objekt verweisen wird

mit synchroner kommunikation & acknowledgements

rekursives freigeben

referenzen von garbage freigeben

rekursiv überprüfen ob nachbarn nun auch garbage

11.6 referenzzähler

für jedes objekt bekannt wie oft darauf referenziert wird

counter aktualisieren bei copy/delete mit inc/dec

kausalität mit synchroner kommunikation / ACK garantieren

zyklischer garbage wird nicht eingesammelt

ACK increment

p sendet inc an r und wartet auf ACK

p sendet copy an q

copy mit verzögerung

lermen/maurer

p sendet inc an r

p sendet copy an q

r sendet ack an q

q darf erst nach empfang ack ein dec auslösen
copy ohne verzögerung
FIFO für $p \rightarrow r$ damit inc vor möglichem dec ankommt

3-nachrichten protokoll (rudalics)

p sendet inc an r
r sendet copy an q
q sendet ack an p
p darf erst nach empfang ack ein dec auslösen
copy mit verzögerung
kein FIFO mehr nötig

4-nachrichten protokoll (rudalics)

p sendet inc an r
p sendet copy an q
r sendet copy an q
q sendet ack nach beiden copy an p
p darf erst nach empfang ack ein dec auslösen
copy ohne verzögerung (aktiv ab erstem empfang)
kein FIFO mehr nötig

11.7 weighted reference counting (WRC)

entspricht konzeptionell kreditmethode bei terminierung
keine verzögerung bei copy/delete; wenige nachrichten

aktionen

"new" setzt reference count rc auf zielknoten
stimmt überein mit reference weight rw auf allen sources
"copy" splittet rw und sendet zu neuer reference source
"delete" sendet eigenes rw an zielknoten
invariante $\sum(rw) = rc$

speicherung rw

im pointer (2bit für bis zu 8 objekten)
als externe int variable
komprimierung 2er potenz wie bei kreditmethode

11.8 local reference counting (LRC)

count pro objekt/core = lokale + remote referenzen
wenn count=0 dann entfernen und dec remote source senden
invariante $\sum(count) \geq$ wirkliche referenzen

bei gleichem objekt / unterschiedliche sources

zB besitzt r referenzen auf objekt o von p und von q
mehrere vorgehensweisen
(a) source-spezifischer counter & benachrichtigung
(b) wenn count=0, alle referenzquellen gleichzeitig benachrichtigen
(c) direkt dec an zusätzliche quellen senden (adoptieren)
bei (c) optimalerweise zu niedrigster indirektion adoptieren

implementierungssicht

einführung von incoming/outgoing reference tables (IRT / ORT)
ORT enthält proxy auf welche interne referenzen (lokal, IRT) zeigen
ORT zeigt auf jeweiliges objekt aus IRT der quelle
IRT zeigt auf objekt, oder wiederum auf ORT proxy
wenn ORT keine referenzen mehr empfängt wird proxy gelöscht
und dec nachricht an IRT der source versandt
lokaler garbage collector mit IRT als wurzel

vergleich zu anderen verfahren

copy auf gleichem core keine zusatznachricht / verzögerung
delete instantan; nur nachricht benötigt wenn count = 0 && remote ref
gleich viele #dec nachrichten wie #copy nachrichten
keine akkumulation gewichtsfragmente wie zB bei WRC nötig
FIFO nicht benötigt
migration objekt zu anderem core einfach durchführbar
aber mehr counter (=in jedem core einer) nötig

11.9 verteilte terminierung \rightarrow garbage collection

problem wird transformiert, nicht algorithmus
umgekehrt auch möglich; aber weniger praktikabel

mit virtuellem objekt R

jeder aktive prozess & nachricht hat referenz auf R
aktive prozesse löschen R (passiv werden)
passive prozesse etablieren R aus nachricht (aktiv werden)

12 wellenalgorithmien

alle prozesse werden am algorithmus beteiligt
wichtiger basisalgorithmus für andere algorithmen

12.1 anwendungen

broadcast information
sammeln verteiler daten; eventl. mit akkumulation
konstruktion spannbaum
phasensynchronisation von prozessen
triggern von ereignissen in prozessen
implementierung schnittlinien (für schnapsschussproblem)
basisalgorithmus (für deadlock, terminierung)

12.2 mögl. voraussetzungen

knotenidentitäten (eindeutig, anonym)
notwendiges wissen (nachbaridentitäten, #knoten)
kommunikationsemantik (sync, async, FIFO, bidirektional)
topologie (ring, baum, graph)

12.3 qualitätseigenschaften

parallelitätsgrad (sequentiell, parallel)
initiatoren (1, 2, viele)
komplexität (zeit/nachrichten, average/worstcase)
bitkomplexität (länge nachrichten)
dezentralität (engpässe)
symmetrie (lokale algorithmen identisch?)
fehlertoleranz (fehlermodelle)
einfachheit (verifizierbarkeit, implementierbarkeit)
skalierbarkeit (kleine/grosse systeme)

12.4 definition

für jede berechnung muss gelten
(1) enthält einziges init und conclude ereignis
(2) alle prozesse besitzen visit ereignis
für alle visit ereignisse gilt
(a) $init \leq visit$
(b) $visit \leq conclude$
für \leq kausalrektion; kausal abhängig oder gleichzeitig

interpretation

information verteilen über init kausalkette (a)
nach conclude wurden alle knoten besucht, terminierung (b)
information einsammeln über conclude kausalkette (b)

schnittlinie

gebildet aus visit ereignissen
trennt menge der ereignisse in vergangenheit/zukunft

12.5 topologiebeispiele

ring

token umlaufen lassen

spannbaum

an blättern reflektieren (wie vereinfachter echo algorithmus)
 $2 * (n-1)$ nachrichten, da immer zwei nachrichten pro kante
idR viele nachrichten parallel unterwegs

stern ("polling")

sequentiell (ein knoten informieren, auf ACK warten, repeat)
parallel (alle gleichzeitig informieren, dann auf alle ACKs warten)

beliebige topologie

möglich mit echo algorithmus (explorer, echo)
visit bei erhalt explorer & senden des echos

12.6 nachrichtenkomplexität

lowerbound

jeder prozess muss mindestens eine nachricht empfangen
da visit kausal von init abhängig sein muss
daher mindestens $n-1$ nachrichten

wenn conclude&init auf gleichem node

nun muss auch initiator eine nachricht empfangen
daher mindestens n nachrichten

ohne nachbaridentitäten

nun muss jede kante durchlaufen werden
daher mindestens e nachrichten

12.7 virtuell gleichzeitiges markieren

FIFO kanäle nötig
mehrere initiatoren möglich

algorithmus

init sends mark to all neighbours
if message received & not marked then
marked = true; send mark to all neighbours
als verbesserung mark nur an unmarked nachbarn senden

auswirkungen

über jede kante läuft in beide richtungen ein marker
schnitt sobald knoten markiert wird

konsistenzbeweis

inkonsistent wenn nachricht schnitt überschreiten würde
dann müsste sender marked und receiver unmarked sein
aber widerspricht konstruktion mit FIFO

12.8 sequentielle traversierungsverfahren

unterklasse wellenalgorithmus mit einzigem initiator
erbringt totale lineare ordnung auf visit ereignissen
auf ungerichteten und zusammenhängenden graphen
token wandert durch alle prozesse & zurück zu initiator
hohe zeitkomplexität da keine parallelität

topologien mit einfachen verfahren

stern (polling; sequentiell oder parallel)
logischer/physischer ring (kreis umlaufen)
baum (depth-first)
n-dimensionales gitter/hyperwürfel (schritt in dim k wenn k-1 fertig)
im allgemeinen alle hamiltonische graphen

12.9 traversierungsverfahren tarry

für ungerichtete, zusammenhängende graphen
besucht alle knoten in max 2e nachrichten

regeln

(r_1) knoten sendet token niemals 2 mal über eine kante
(r_2) knoten sendet token über aktivierungskante zurück wenn alle kanten besucht

beweis

(1) terminiert beim initiator
für alle anderen knoten gilt
hat p das token, wurde es k-mal erhalten & k-1 mal versandt
daher immer noch kanal frei um auszusenden
⇒ somit bleibt token nicht bei p
(2) kanäle initiator in beiden richtungen jeweils einmal benutzt
alle ausgangskanäle benutzt gemäss (r_1)
muss genauso oft zurückgekehrt sein gemäss (1)
rückkehr nicht zweimal über den gleich kanal gemäss (r_1)
(3) kanäle in beiden richtungen jeweils einmal benutzt
(beweis durch widerspruch)
sei p erster knoten für den dies nicht gilt
sei vater(p) dessen vorgänger (somit bedingung erfüllt)
damit hat p über aktivierungskante versendet (r_2)
das macht p aber erst nachdem alle anderen besucht wurden (r_2)
somit muss p aber auch n-mal das token erhalten haben
somit kann die eigenschaft für p nicht stimmen
(4) alle prozesse wurden durchsucht
da graph zusammenhängend müsste es kante geben von besucht → unbesucht
besucher prozess nutzt aber alle kanäle gemäss (3)

vergleich depth-first

spezialfall tarry; (r_1), (r_2) gleich aber zusätzlich (s_3)
(s_3) token kehrt um, wenn knoten bereits besucht
dadurch entstehen deterministische spannbäume
schliesst kanten aus die gleiche levels verbinden
tarry um (r_3) ergänzen damit bereits besuchte knoten direkt zurücksenden
(r_3) sofern (r_1) und (r_2) erfüllt wird token direkt zurückgesendet

12.10 traversierungsverfahren awerbuch

bei unbekannten nachbaridentitäten
parallele hilfsnachrichten während sequentiellem tokendurchlauf
"visit" nachricht erklärt dass token bereits knoten besucht hat

algorithmus

neuer empfänger auswählen (noch kein visit erhalten)
visit an alle ausser aktivierungsknoten, empfänger
auf acks warten (weil nachrichtentransferzeit unbekannt)
dann token an tokenempfänger weiterleiten
in 4n-2 zeit, 4e nachrichten (2 visit, 2 ack)

verbesserung cidon

keine ACKs für die visits; direkt weiterversenden
falls token $q \rightarrow p$ und gleichzeitig visit $p \rightarrow q$
dann interpretiert p token als visit, q generiert neues token
in 2n-2 zeit, 4e nachrichten (2 visit, 2 token)

13 schnappschuss

13.1 anwendungen

analytics wie lastverteilung, geld im umlauf bankensystem
globale eigenschaft wie deadlock, terminierung, garbage
konsistenter zustand verteilter datenbanken

13.2 herausforderungen

prozesszustände nur nacheinander erfahrbar
gleichzeitigkeit unmöglich da unbestimmte nachrichtenlaufzeit
zustandsrelevante nachrichten möglicherweise in transit
ermittelter zustand veraltet & mögl. inkonsistent

13.3 resultateigenschaften

konsistent (konsistente schnittlinie)
möglich (zustand wäre möglich gewesen)
vergangen (zeitverzögerung nicht verhinderbar)

13.4 triviale lösungen

system anhalten und zählen (wie volkszählung, inventur)
synchronisierte lokale schnappschüsse

13.5 schnappschussalgorithmus

wellenalgorithmus um node von schwarz zu rot färben
beliebige wiederholung mit weiteren farben

rot werden (schnappschussmoment)

durch empfang roter nachricht
durch besuch des wellenalgorithmus
lokaler zustand initiator melden

rekonstruktion zustand

schwarze nachrichten nach rot werden an initiator weiterleiten
beim schnappschussmoment initiator auch #sent/#received melden
initiator wartet bis alle schwarzen nachrichten empfangen

13.6 chandy/lamport

braucht FIFO kanäle
farbe muss nicht mehr in nachrichten mitgeführt werden
konsistent indem keine nachrichten von rot → schwarz
globaler zustand besteht aus prozess & kanalzuständen

marker-sending rule

prozess p wird "markiert" & zeichnet eigener zustand auf
vor versand weiterer nachrichten marker über alle channels

marker-receiver rule

zustand c = nachrichten nach markierung q & vor empfang marker c
wenn q noch nicht markiert marker-sending rule ausführen

eigenschaften

zustand c leer wo markierung empfangen wird (spannbaum)
mehrere initiatoren möglich

14 logische zeit

gesucht sind logische zeitstempel zur aussage über kausalität
für abbildung $C: E \rightarrow H$ (ereignis → zeit) soll $e' < e \Rightarrow C(e) < C(e')$
falls möglich soll umkehrung / weiteres mit C möglich sein

14.1 anwendungen

debugging (ursachenfindung, replay, synchronisationsfehler)
performance (unabhängige ereignisse gleichzeitig ausführen)
implementierung kausaltreues broadcast/order
implementierung konsistenter schnappschüsse

14.2 lamport

lokale uhr tickt bei jedem ereignis
mit nachricht wird lokale uhr mitversandt (nach increment)

empfangsereignis $\max(\text{lokale uhr, empfangen})$, dann incr
aber strukturverlust gleichzeitigkeit

E→H

wie erwartet

$a < b \Rightarrow C(a) < C(b)$

$a > b \Rightarrow C(a) > C(b)$

$a \parallel b \Rightarrow \text{relation } C(a) \text{ zu } C(b) \text{ beliebig } (>, <, =)$

weil E halbordnung, aber $(<, \parallel, >)$ linear

H→E

gleichzeitigkeit immer möglich

$C(a) = C(b) \Rightarrow a \parallel b$

$C(a) < C(b) \Rightarrow e < e' \text{ oder } e \parallel e'$

$C(a) > C(b) \Rightarrow e > e' \text{ oder } e \parallel e'$

kausal unabhängig wird somit mögl. kausal abhängig abgebildet

14.3 vektorzeit

zeitstempel als n-dimensionaler vektor

repräsentiert gesamte kausale vergangenheit zum schnittzeitpunkt

eintrag ist jüngstes lokales ereignis des jeweiligen prozesses

operationen

\leq wenn komponentenweise \leq ; sonst \parallel

$\sup(a,b)$ = komponentenweises maximum

implementierung

beim senden zeitvektor mitsenden

bei empfang $\text{sub}(\text{lokal, empfangen})$ erstellen

bei jedem ereignis eigener eintrag erhöhen

schlussfolgerungen

$e < e' \Leftrightarrow t(e) < t(e')$ (kausalkette ist vorhanden)

$e \parallel e' \Leftrightarrow \text{not}(t(e) < t(e')) \wedge \text{not}(t(e) > t(e'))$

damit limitierung lamport zeit aufgehoben

idealisierter beobachter id(e)

nimmt ticken aller prozesse global sofort wahr; in $\text{id}(e)$

es gilt für alle beobachtungen $t(e) \leq \text{id}(e)$

$t(e)$ kann zu $\text{id}(e)$ gummibandtransformiert werden

14.4 schnittmatrix

columns jeweils zeitvektor prozess während schnittereignis

konsistent wenn $\text{dia}(\$) = \sup(\$)$ (diagonalvektor = supremum columns)

intuitiv damit niemand aktuellerer stand als lokaler prozess hat

beispiel inkonsistenz

besuch P_1 zur zeit z_1

P_1 sendet zu z_2 eine nachricht an P_2

besuch P_2 zur zeit z_3

nun hat P_2 im zeitvektor z_2 vom P_1

im diagonalvektor steht bei P_1 aber z_1

daher $\text{dia}(\$) > \text{sub}(\$)$, somit inkonsistent

konsistenter schnappschuss

alle prozesse senden lokaler zeitvektor zu beobachter

falls beobachter konsistente matrix hat, terminieren

andernfalls wiederholen

kausaltreuer beobachter

ordnung nachrichten damit $\text{dia}(\$) = \text{sub}(\$)$ immer erfüllt

beobachter muss sich dazu nur diagonalvektor merken

kausale broadcasts

verzögerung verarbeitung nachricht bis vorherige eingetroffen sind

nützlich zur implementierung von FIFO