

Система, предност, наиме, основните особини, middleware, поден +  
ТАНЕЦИ БЛУМОВА КНИГА ПОГЛАВЛЕ 1 - Introduction

Дистрибуирани системи - ТЕОРИЈА - ЧВОД [01]

\* развој DC - појава микропроцесори и брзе мреже  
\* дефиниција DC - "A collection of independent  
computers that appears to its users as a  
single coherent system", A. Таневски

\* одредности DC: НАД ЦЕНТРАЛИЗОВАНИМ: (+)

1) Економија	1) Економија
2) Брзина	2) Брзина
3) Погодност	3) Погодност
4) интегрираност	4) интегрираност
5) Деловне ресурси	5) Деловне ресурси
6) комуникација	6) комуникација
7) Едикатно коришт. ресурс	7) Едикатно коришт. ресурс
	8) Едикатно коришт. ресурс

\* НАИМЕ DC у односу на централизоване: (3)

1) ВЕЋА КОМПЛЕКСНОСТ - hw, DC, програмски језици  
2) УЧИРДЕЛСВАЊЕ 3) БЕЗБЕДНОСТ (због приступности)  
4) ОСНОВНИТЕ ОСОБИНЕ DC: (4)

1) ХЕТЕРОГЕНОСТ 2) ТРАНСПАРЕНТНОСТ 3) Отвореност

4) СЛАБИЛНОСТ (3) → решење је OSI модел (4 прасци)  
5) ХЕТЕРОГЕНОСТ - hw (ISA), DC (Socks), RJ (интегрирана)

6) ТРАНСПАРЕНТНОСТ - "скривати" сложак DC

=) типови транспарентности: (+)

1. приступна	1. приступна
2. локацијска	2. локацијска
3. миграционта	3. миграционта
4. конкуренција	4. конкуренција
5. репликација	5. репликација
6. за отлаз	6. за отлаз
7. паралелизација	7. паралелизација

7. за отлаз

\* Отвореност - компоненте DC се тако могу користити  
од стране других система или се интегрираат у

## \* Скалабилност (проширливост) - въз з лимити (3)

- 1) број користещи и ресурса
- 2) ГЕОГ. УДАЛОСТ користещи и ресурса
- 3) број търговски активни облака просират на (4)

=> Експертите със ~~ДЕЦЕНТРАЛИЗОВАНИЯ АЛГОРИДМА~~<sup>10</sup> НЕ ИМАЮТ ИНФ.

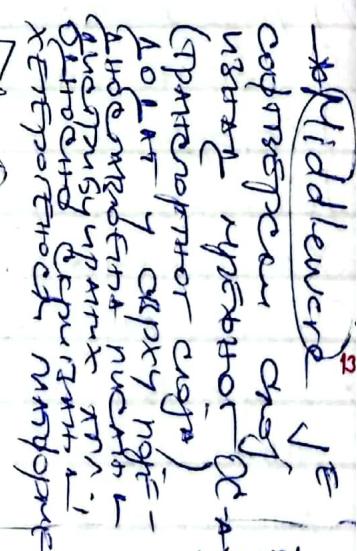
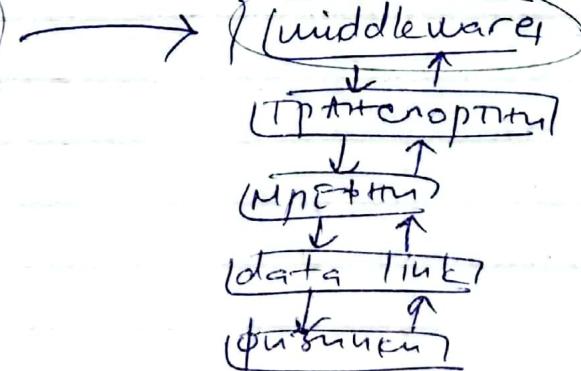
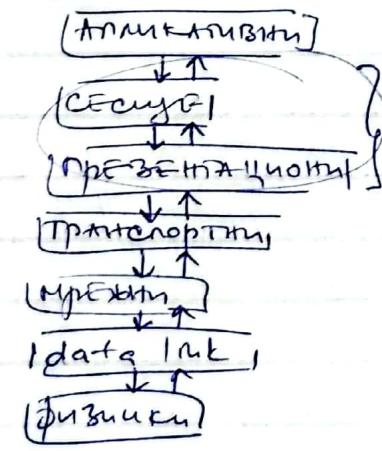
- |  |  |
|--|--|
| 1) <del>ИИЕДНА МАШИНА НЕ ЗНАЕ КОМПЛЕКСНО СЪЩЕСТВУЮЩИ</del> | 1) <del>ИИЕДНА МАШИНА НЕ ЗНАЕ КОМПЛЕКСНО СЪЩЕСТВУЮЩИ</del> |
| 2) <del>МАШИНА ДОНОСЕ ОДИЧУЩИ РЕШЕНИЯ</del>                | 2) <del>КАКО НА ОСНОВА ПОСЛЕДНИХ ИНФ.</del>                |
| 3) <del>ОТКАЗ И ЕДНА МАШИНА НЕ НАРУШИВА СИСТЕМА</del>      | 3) <del>ОТКАЗ И ЕДНА МАШИНА НЕ НАРУШИВА СИСТЕМА</del>      |
| 4) <del>НЕМА ПРЕПОСТАВЛЕНИЯ О ПОСЛЕДНИИ</del>              | 4) <del>НЕМА ПРЕПОСТАВЛЕНИЯ О ПОСЛЕДНИИ</del>              |
| <del>последните резултати</del>                            | <del>ЧАСОВИКИ</del>  |

## => ТЕХНИКЕ СКАЛПРИЯТ: (3)

- |                            |  |
|----------------------------|--|
| 1) Скалиране ком. траф.    | 1) Скалиране том. бандвдл              |
| 1.1) Аутентризиране        | 1.1.) Аутентризиране                   |
| 1.2) download-изтегляне    | 1.2.) Създаване на сървърска структура |
| 1.3) Дела сървърското къде |  |
| 2) Дистрибуция             | 2) Дистрибуция                         |
| 3) репликација             | 3) репликација                         |

=> Скаларите може да използват ~~НЕСОВЪЗСЪЕДИНАЕМОСТ~~

## \* Изменени OSLI referentni модели:



\* Типови middleware - RPC, RMI, Message oriented (MPI, MSGQ, Pub/Sub)

## \* Типови AC

I на архитектурном ниво

1) client-server

2) peer 2 peer

3) управляване (controlled),  
специфично

II ЧОЛОСИ ИЗ ОБЛАСТ  
ПРИМЕР

1) дистрибуиран рач. сис.

1. a) класери

1. b) грип

2) дистр. информационни сис.

2. a) сис. за обработка транзакции

2. b) сис. за интеграция пословни  
търговски

БРС-1 (Белое пятно с параллельными линиями, симметрическим промежутком, без РРС, ДС и РС  
стабилизатора) и поправка 1.2

Гипотеза оценки - ТЕОРИЯ - Результаты + 1 - [02]

Дистрибуции  
и "революционары" идея - дозволить профессии да  
извий профессии на ученый институт - RPC  
(3+1)

1. проект парламентария

2. хевропроект - кредитка и сертификат

3. личное участие МАИМОНА САИМПА.

4. семинар предпринимательства

5. проект парламентария

1. Намирати претпоставки за изпълнение на проекта
  2. Известяват съдържанието на проекта
  3. Извършват изпити на проекта

## 1. Семинар презентация

4. Семантический преобразователь параметров: (3)

1. по временному 2. по референции 3. by "Copy/restore"

\* появляется клиент и сервер (client process, server process)

```

graph TD
    subgraph Client [Client]
        direction TB
        CP[Client program] -- (10) --> LS[Local stub]
        LS -- (11) --> N[Network]
    end
    subgraph Server [Server]
        direction TB
        SP[Server program] -- (1) --> SS[Server stub]
        SS -- (12) --> LR[Local routines]
    end
    N -- (3) --> LR
    LS <-- (8) --> N
    SS <-- (7) --> LR
    LS <-- (9) --> SS
    LS <-- (1) --> SP
    SP -- (6) --> SS
    SP -- (5) --> LR
    
```

1. Клиент вызывает локальную процедуру (4 + СПС) (10)

2. Клиент stub marshalls аргументы и вызовы при помощи локального OS-а за счет порука через сеть

3. Локальный OS шлюзует поруки и просматривает серверским stub-ом

local kernel  
remote kernel  
local kernel  
remote kernel  
локальный порт к TCP  
запрос

1. ПОЧЕВА  
4. УЛАНЕТИМ OS ПРИМА ПОРУКЕ, РАДИ UNMARSHALLING  
АРГУМЕНТАЗА ПОЗИВ ПРОЦЕДУРЕ  
5. СЕРВЕРСКИ SAMB ПОЗИВА ХЕНДЕЛЮ СЕРВЕРОСУ ПРОЦЕ-  
ДУРУ И ПРЕДЛЖЕ ЈОЈ ПАРАМЕТРЕ ПРИМИЛЕТЕ ОГ СРАТНЕ  
СЛУЖБЕНТА  
6. СЕРВЕР ИЗВРШИЛА ВА ПОЗИВАНИ ПРОЦЕДУРУ И ВРАЋА  
7. СЕРВЕРСКИ SAMB MARSHALL-УЈЕ РЕЗУЛТАТ И ПОЗИВ  
ПРИМИЛВЕ OS-А ЗА СРАТНЕ ПОРУКА КРОЗ КУДУК  
8. УЛАНЕТИМ OS ЧИНАЕ ПОРУКЕ ЛОКАЛНОМ OS-У  
9. ЛОКАЛНОМ OS ПРОСЛЕДИЈЕ ПОРУКЕ КУДЕНТИСКОМ SAMB  
10. КУДЕНТИСКИ SAMB РАДИ UNMARSHALLING И ВРАЋА  
РЕЗУЛТАТ ПРОГРАМУ КОЈИ ЈЕ ПОЗВАО ПРОЦЕДУРУ

→ пријава проблема

(2)

→ претпоставка да је RPC - по вредностима и структуром

→ Код који подизирају услугом сервер и процесору (2)

⇒ финално → 1) статичко повикавање - мапирају услуге, /  
2) динамично повикавање - сервер има имена /

→ Статичка позивка (програм) RPC - (2)

1) Бар једном (имплементације) 2) Гашно једном

→ појам IDL - Interface Definition Language

⇒ компонира се помоћу IDL/RPC компоненте

→ Сим RPC + XDR и грејден, пројектован за Sun OS

⇒ компонирањем XDR спеч., грејден генератор: (3)

1) header фајл (primer.h) - јединствени ид. интерфејса

дефиниције типова, константи и прототипова функција

2) клиент јављ (primer-clnt.c) - садржи процедуре

које НЕ бивају позивати; процедуре су задужене за

постављање, слатије, прузам и расподавање порука резултата

3) сервер јављ (primer-svc.c) - садржи процедуре

које се позивају када порука стигне до сервера

(са којим се среће) и које затим позивају одговарајуће

серверске процедуре.

⇒ дефиниција интерфејса - писање XDR спецификација

- састоји се од 2 делова - XDR дефиниције и дефиниције

⇒ први део XDR фајла садржи дефиниције типова

и константи + други дефиниције процедуре

⇒ свака процедура је јединствено описане

брдем програма, бројем вршића и бројем процедур

има један ул. и један изл. параметар; у клијентском

коду УВАЛДО. проп. се позивају са impreocedure-brverize;

а у серверском имају и јОЛАТАК - src;

⇒ програмер писаје XDR спецификацију, клијентски је

тада и имплементираје услуге које процедуре

⇒ грејден (port-mapper у оквиру SunRPC-а) ослушије

нападаваче које су на порту 111 и одржавају

динамичку базу доступних RPC сервиса

(program, version, protocol, port.)

1) Имета би било производивно да се именувају  
 • ЕГР 1, речимо AVG пример, где би се искористило  
 ово што је поменуто на презентацији

\* Битни позитви на клиентској странци:  
`clnt=clnt-create(server, KALK, KALK-VERSION, "udp");`  
`zmr=saberi(&op, clnt);` (RPC-ЕВИ)  
`razlika=oduzmi(&op, clnt);`  
`clnt-destroy(clnt);` (РАСЧЕЛЯЊЕ КОНЕЦИЈА)

\* Битни програм-ови:  
`rpcgen -S printer.x`, `rpcgen -C -Ss printer.x > Server.c`  
`*.h * clnt.c * -Sv.c` (Server.c)

### БРОДНОСТ САМ RPC → (3)

- 1) Али. НЕ мора да воли рач.
  - 2) Али. НЕ мора да воли рач.
  - 3) Али. мора да знае само колко ће број порта
1. НЕ ВОЛИ РАЧУНАР  
О ЈЕДНОСТАВНОМ ПОРТУ
2. ВЕЛИЧИНА ПОРЕД ДРГИХ  
И ПРЕСЕЛБИРЉАВА
3. САМО ТРИ ГЛАВНИХ ПОРТА  
СЕГДА је грешка

### DCE → Distributed Computing Environment

⇒ БАЗИРАН НА КЛИЕНТ-СЕРВЕР моделу

### DCE сервиси: (5)

- |                           |                   |
|---------------------------|-------------------|
| 1) RPC                    | 1) RPC            |
| 2) Директоријуски         | 2) Директоријуски |
| 3) дисп. фул. сервис      | 3) ДФС            |
| 4) БЕЗБЕДНОСТИ            | 4) БЕЗБЕДНОСТИ    |
| 5) дистрибуиратор времена | 5) дисп. времена  |

⇒ DCE RPC - концепција, платформа за све остале

⇒ DCE CDS - коришћење копијских имена у оквиру

⇒ БЕЗБЕДНОСТИ → аутомултација и аутентификација

⇒ Општи часовник на разн. машинама

⇒ IDL - Код. АЛНО јЕДНОСТАВНИ УЛТИФИКАТОР Интерф.

⇒ midgen metafile.idl → подредио је основни IDEPENDENT

⇒ ОБЕЗБЕЂЕНЕ су "БАР ЖЕДНОМ" и "САМО ЖЕДНОМ" СЕМ.

⇒ idl printer.idl → DCE програмер пише,

1) клиент код 2) сервер иницијализација 3) аутентификација

2. код који се региструје, користијући експорт

⇒ rpcd - порт мапер

Дистрибутивни архитектура - Концепција 2. [Ф3]

Клиенте и сервере - објекти са адресом contract - S<sub>contract</sub>

Управљачки објекти се може приступат једино

RMI - Remote Method Invocation - proxy и скелетон

Су апликацији client и сервер server - proxy - проследитељи

\* Референција удаљеним објектима - скелетон се од

неколико поља - IP, port, time, #obj, interface info

\* Java RMI - удаљени интерфејс, удаљени објекти,

референција удаљеним објектима, finder (некојој

имају са серверне имена) RMI Registry

\* Потезавање клијента и сервера: (6)

1. Сервер региструје удаљени објекат у реестру finder под одређеним именом - удаљеним објектим
  2. Клијент поводом апликација хешем да приступи удаљеном објекту, онја користи истово име да да би од finder-а запретио референцију помоћу које иницијира proxy (клијент везе)
  3. Клијент позива метод удаљеног објекта, и позив се проследије proxy-у
  4. proxy се информише о информацији потребите за позив и штави их скелетону кроз пребар
  5. скелетон прима и десеријализује поруку, позива одговарајући метод, резултат се резултат, врши се серијализација и слање
  6. proxy прима одговор, десеријализује и врши дејући клијенту
- => свих 6 корака су за клијента потпуно прати спаренити
- \* java.rmi.RemoteInterface, RemoteException,  
java.io.Serializable, UnicastRemoteObject,  
Naming.rebind, LocateRegistry.createRegistry
- \* преткос напомена - по вредностима (сопствените) и по референцији (remote)

- \* Типови комуникација - параметри поделе (3)
  - 1) нервистичност - нервистичне / транзитентне
  - 2) синхронизација - синхронне / асинхронне
  - 3) вртежност - дисперзна / стримована
- \* нервистичност - порука се памти у ком. серверу
  - ако неко комад је потребно да би се испоручило <sup>уредништво</sup>
  - транзитентне - чекајуко НЕСИ ОД ком. сервера <sup>увиједом</sup> да приђе поруку, она се одбације; ком. сервери су као store&forward ~~буфер~~ рутери
- \* асинхронн - почињати настављати одмах насон сланца поруке, НЕМА блокирања је порука се памти или у локалном буфери (middleware) или у ком. ср.
- \* синхронне<sup>12</sup> - почињати је блокирати СВЕ док се НЕ потврди прихватање његовог захтева;

#### ЕТАПНИТИВЕ ЗА БЛОКИРАЊЕ: (3)

- 1) док middleware НЕ преузме руковање захт.
- 2) док порука НЕ синхре до примача
- 3) док порука НУЖЕ потпуно обрађена и примача НЕ врати резултат, односно одговор

\* дисперзна - свака порука садржи комплетан <sup>скуп</sup> информација

\* стримована<sup>14</sup> - сланце више порука заједно, у врежинском односу или повезаних редоследом сланц, што је потребно за реконструкцију комплетних информација

\* синхронне и асинхронне комуникације <sup>типови</sup> (1)

- 1) нервистичне синхронне
- 2) нервистичне асинхронне
- 3) транзитентне синхронне
- 4) транзитентне асинхронне

\* нервистичне асинхронне - порука се памти у буфери локалног хоста или првом ком. серверу

\* нервистичне синхронне - почињати је блокирати док се порука НЕ запамти у одредином хосту

\* Асинхронните извършители - порука се пригответе  
пакет у локалния буфер извора и док тя  
изпраща са извршенето, middleware просле-  
дява портука  $\Delta$ кое;

\* Асинхронните извършители - общици (3)

1) Блокиране извора док се приема  $\text{НЕ ЗА-}$   
пакет у буфера  $\text{записаното}$  хоста ( $\text{записаното}$  ACK)

2) Блокиране извора док определени процес  
не приема порука (определите генеричен ACK)

3) Блокиране извора док определени процес  
не генерира и връща резултат

\* Middleware базирани на съмните порука

$\Rightarrow$  MOM-Messagе Oriented Middleware

\* MPI (Message Passing Interface) - поддръжава

СВЕ типове асинхронни комуникации

$\Rightarrow$  поддръжава P2P и групни комуникации

$\Rightarrow$  членство на група се определя  
(group-id, process\_id) за членстване

$\Rightarrow$  поддръжава членство комуникации (8)

1) MPI\_Bsend - Асинхронна асинхронност

2) MPI\_Ssend - Синхронна асинхронност, тип АА 2

3) MPI\_Send - --, тип 1

4) MPI\_Sendrecv - --, тип 3

5) MPI\_Isend - проследи референту и настави

6) MPI\_Issend - проследи реф. и чекай док не бъде

7) MPI\_Peek - Блокира се док не приемиш порука

8) MPI\_Irecv - ~~Блокира~~ чете са приемом, или без блок

\* MQS (Message Queueing System) - перзистентни

асинхронни комуникации, користейки редова чекана

и конфигуриране - оба неактивни, оба активни,

които имат активни, приемани активни

\*~~ПРЕСУРГАЦИЈЕ~~<sup>23</sup> СЕ ОВАДНОУ ПРЕСУРГАЦИЈАМ ВРЕДИЧНОГ РЕДА  
\*~~ОПЕРАЦИЈЕ~~: (4)

- 1) put - стављање поруке у ред
- 2) get - читanje поруке из реда, са блокирањем
- 3) poll - читanje поруке из реда без блокирања
- 4) notify - иницијализација handler-а за осигурује  
делије прискон пренос поруке

\*ПОЛИТИКА УПРАВЉАЊА РЕДОМ је FIFO и не придржи  
\*MQS подржавају P2P<sup>24</sup> модел размене порука

=> само JEDI AN ПРИДУМАЛА НА САМО JEDI НОГ  
ПРИМАЊА ЧУ РАСПЕЦИЈАЛНЕ ПОРУКЕ, И ЧУ  
ПРЕД ЈЕДНОГ РЕДА МОГУ БУДИ ПОДВЕЗАНИ MQS  
\*ПРЕДЛОЖИЋА УПРАВЉАЊУ УПРАВЉАЊУ РЕДОВА - QM  
(Queue Manager) и то најчешће ЧУ конфигурацију  
ДЕСУ ОВА ПРОЦЕСА НА ИСКОН/ХОСТУ

\*~~ПРИДУМАЛА~~ РЕДОВА ЗА КОНФИГУРАЦИЈУ: (3)

- 1) КОД СУ ПРЕСУРГАТИ РЕДОВИ? — логичка  
имена, али морају да поседују пресикатавање
- 2) ДОСТУПНОСТ ПРЕСИКАВАЊА УПРАВЉАЊИМА?

=> JASER+ ПРЕСИКАВАЊА, ПОВОДИ ДО ТРОБЛЕНА  
ОДРЖАВАЊА СОЛДАВАЊА НОВИХ РЕДОВА  
3) ЕДИТАСТРО ОДРЖАВАЊЕ? => посебним QM који саку  
КАО РУТЕРС, само прослушују поруке ЛИДЕ;

ОД САМО ТРЕБА ДА ЗНАЈУ ТОМ СУСЕДУ ТРЕБА +  
ПРОСЛЕДИТИ ПОРУКУ

Топологију уреди, а обичан QM  
трећи саку да биде

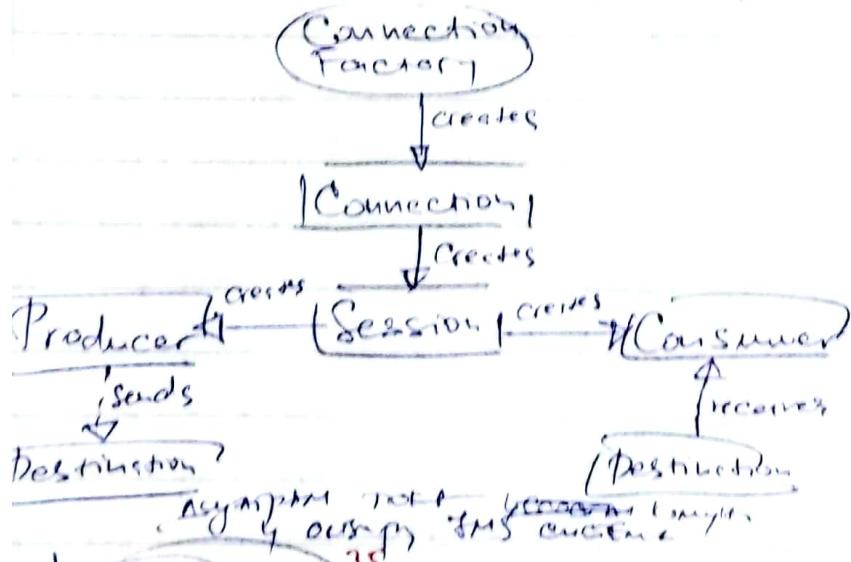
\*~~PUBLISH/SUBSCRIBE~~<sup>26</sup> СИСТЕМ ЗА РАЗМЕТУ ПОРУКА

=> publisher = примилац, subscriber = прималац,  
topic ≡ теме; сваки subscriber-а може прими  
и поруку објављену од сваког jefnog publish-

er-а, за размаку од MQS+

=> ПРЕДЛОЖИЋА НА ТЕМУ (Topic) морају бити са  
ним без филтера

- ⇒ фильтрирање на основу теме
- ⇒ subscriber-и се могу претплатити на тему или на основу адреса<sup>30</sup> (адресни филтер) како службен интерфейс<sup>31</sup>:
  - 1) publish(event) - објавување додатка
  - 2) subscribe(filter) - претплата на основу теме
  - 3) notify(event) - ханделер за обработка додатка
  - 4) unsubscribe(filter) - прекид претплате
- ⇒ JMS (Java Message Service)
  - ⇒ поддршка MQS и Publish/Subscribe (на основа Topic-а)
  - ⇒ JMS клиент<sup>32</sup> - производител на неки поддршак интерфејс
  - ⇒ JMS провайдер - иплементира JMS интерфејс.
  - ⇒ JMS порука - објект кој имплементира Message класа
  - ⇒ JMS дестинација - Topic или Queue
  - ⇒ JNDI (Java Name Dir. Int.) - службички како наше S.
  - ⇒ Connection factory - чистарка конфигурирана од страна администратора, служи како "нос" као JMS производеру, експонују функционалност најакот сервис и нуди једнократни итератор



\* Approfe meadow  
A no yesterdays  
consequently as this  
proposition who  
are about my new  
disorder continuing  
tomorrow will be  
very disagreeable not  
from that topic we  
have discussed

~~БЮДЖЕТ~~ <sup>35</sup> CE КОМПАНИЯ ЕСТЬ НА  
ВАШЕ ~~ДЕДУХИ~~, <sup>36</sup> CE СЕДЬЯ ПРЕДСТАВЛЯЕТ  
КОМПАНИИ ЗАЩИЩЕНЫ СВОИ ОНЛАЙН-ОБЪЕКТЫ <sup>37</sup> CE КОМПАНИИ  
CE МЕНДРАНС ЗА FMC-НОМЫ ИМЕЕТ CE КОМПАНИИ  
~~ПРОИЗВОДСТВОМ, ПОСЛОДИСМ, ИДОУ И САДЫБО~~

портуза се сметају да је део - ~~Систем~~ Система објеката и тима порука

\* финални поруци који су MSG (a)

=> Message, TextMessage, StreamMessage, MapMessage, ObjectMessage, BytesMessage

=> Message, TextMessage, StreamMessage, MapMessage, ObjectMessage, BytesMessage

бронзови - пренос и отпремајуће поруке на општујућу реч, чиме је одређеној групи

\* поточни - определене поруке, вакуумизација ~~настала~~ намењена именованом систему custom listener-a

обични класи који имплементирају интерфес

\* Queue и Topic су објекти који се користе за преносима информација, бити веома употребљавани

\* R2P може да QueueConnectionFactory, QueueConnectionQueueSession, QueueSender, QueueReceiver

\* Pub/Sub = TopicConnectionFactory, TopicConnection, TopicSession, TopicPublisher, TopicSubscriber

=> време, пријатељи часовници, алгоритам за уче сушта. физ. час

дистрибуирати системи - Европа - Синхронизација 1 Год

\* те постоји појам тоболиног времена у оквиру AC, што је одржавање конзистентности стварајући

тиме проблем. Европа спуштајући њих потребно посматрајући консултантог времена, само \* синхронизација

међу дистрибуиратим процесима (због конкуренције и одјеване компјутерске памћење)

\* Алијорт је увек појам који сак часовниција којима се решава проблем, синхронизације

\* Физички часовник - бројни и хладнији резистори и алгоритам за синхронизацију часовника!

=> Криолошка алгоритам - Европа синхронизација - систем који једну минутицу да WWR часовником (тзв. сервер времена) и њега потпуји сви осталите

часовници - уједно да синхронизују све њихове часовнице



$\Rightarrow$  по Кристјантовом алг.:  $T_{new} = C_{out} + \frac{T_1 - T_0}{2}$

али чако је добуено време "МАЊЕ" од времена, часовник НЕ СМЕ! да иде у назад, већ овај проблем мора да се решити успоравањем локалног часовника

$\Rightarrow$  проблеми Кристјантовог алгоритма: (3)

- 1) ВРЕМЕ НЕ СМЕ да иде у назад
  - 2) Ако је  $C_{out}$  "МАЊЕ" од времена локалног часовника, онда једноставно преузимање времена нисе могуће, већ локални часовник трета да се успори
  - 3) пропадајући кашњење-бележе се  $T_0$  и  $T_1$ , па се ново време рачунат, узимајући их у обзир  
~~Berkeley UNIX~~ Алгоритам има супротан прилаз - сервер имена није пасиван као код Кристјантовог алгоритма, већ активно ~~пакет~~ обавља прозивку свих рачунара на првом и другом нивоу синхронизације - метод је погодан за системе који немају WAN
- Пријемник

$\Rightarrow$  Кристјантов алг. и Berkeley UNIX алг. су првостепено професионални за синхронизацију часовника у LAN-у, односно машинама и ~~предавачима~~ NTP (Network Time Protocol) је дизајниран за синхронизацију уређаја на WAN нивоу, односно за сам Internet - то је протокол апликационог нивоа који користи хипервртјес ~~предавачи~~, уређење сервера времена за синоним физичких часовника машине везаних на Internet

$\Rightarrow$  Срвери су извори информација о тачном времену; на транспортном нивоу користи се UDP срвери су организованы у TBS. ~~Стратем~~ 6

Односно UDP

=> постоје 3 ревима за синхронизацију часовнице у часовници: (3)

1) симетрични ревим - најпредизитнији, користи се за синхронизацију мастер Сервера

2) клиент-сервер (RPC) ревим - слично као Крипцјатов алгоритам  $\leftarrow$  речи?

3) multicast ревим - користи се у Брзим LAN-импортоვе маркете, већима часовници, алгоритам узимајући исказувач алогоритам

## Дистрибујући систем-Теорија-Синхронизација 2 [POS]

\* то је важно физичко /абсолутно време слат је дојдјају наступно, већ релативни редослед дојдјају релативно уредење дојдјаја се постоеће ТВР-логичким часовницима

\* Алиорт је дефинисао релацију если се пре односно "happened before"  $A \rightarrow B$  ( $A$  се десило пре  $B$ )

=> ова релација је увек тачна у 2 случаја:

1)  $A$  и  $B$  су дојдјаји у оквиру истог процеса

2)  $A$  је слате,  $A \rightarrow B$  пријем поруке

=> ова релација има особиту транзитивност

\* Алиортови логички часовници подјирати су, у middleware-у, између апликативног и преносног

\* Ако важи  $A \rightarrow B$ , тада  $T(A) < T(B)$

1) правила Алиортовог алгоритма: (3)

1) логички часовник  $L_i$  се инкрементира при извршењу блок које ажурира у процесу  $P_i$

2) када  $P_i$  шале поруку  $m$ , онда ће увадије

и време слата,  $t = L_i$

3) када процес  $P_j$  прими поруку  $(m, t)$ , онда

т одређује  $L_j$  као  $\max(L_j, t) + 1$ , а затим је прослављаје апликативном слоју

2) проблем - могуће је да више конкурентија дојдјаја и дају исте временске маркете

⇒ Уређење и приказнице да свака мрежница бије единствена - мрежница поседује  $(T_i, i)$ , где  $i$  је број процеса (јединица)

⇒  $(T_i, i) \leq (T_j, j)$  односно  $T_i \leq T_j$ , или  $T_i = T_j \text{ и } i \leq j$

⇒ Потпуно уређење приступа комутацији =

примена имплементације мрежице

\* Всекорасни чланови мреже на основу имплементације мрежице не може се усредновити који додавачи су неизбјегљиво усвојени - а који конкурирају - помоћу всекорасних члановника, односно тиховим упоређивањем, након добити ову информацију

⇒ Правила алгоритма синхронизације помоћу

всекорасних члановника:

1) Всекорад се иницијализује  $V = \emptyset$  и свим процесима (већина в-ора = број процеса)

2) Процес  $P_i$  имплементира  $i$ -ти елемент всекорад пре сваког додаваја, тј.  $V[i] = \emptyset$

и увек што је порука виједно да својим в-ом прихвати  $P_j$  прими поруку, пореди локални в-ор са првиотним поткомпонентом и постави в-ор  $E_k$ . Уколико сваку да вију вр.

ако  $V[i] = V'[i]$  за све  $i = 1, n$ ,

тада  $V = V'$ , а ако  $V[i] \leq V'[i]$  тада

важи  $V \geq V'$ , реално  $V \leq V'$ ; уколико ће вију  $V \geq V'$ , иако  $V \leq V'$ , тада су  $V$  и  $V'$  уједињене

⇒ Потпуно уређење multicast комутација може се имплементирати и помоћу

всекорасних члановника, с тим што су правила мало комплетнија:

1)  $V[i] = V[i+1]$       2)  $V[k] \leq V[j], \forall k \neq i$

Четврти алгоритам чврзаног ислаучиштава: (2)  
1) Центризовани 2) дистрибуирани 3) базирани на жетонима  
Центризовани АЛГ. Један процес је одговоран  
 за координатора; ~~који~~<sup>12.</sup> процес има да при-  
 супи довољном ресурсу, што је захтев соорги-  
 натору; ако тај даји процес тије у критичкој  
 скену, ~~који~~ координатор "започео" KC, ако  
 потврди; ~~у тој~~ уколико је ке запушта, за-  
 мада се смета у ред некада  
=> КАДА: координатор усвоји грешку; ~~и~~ ако  
 координатор откаже; ако може се дете-  
 ловати отказ координатора

ПРЕДНОСТИ: лак за иплементацију;  
 захтева разлику само 3 поруке; фер;  
дистрибуирани АЛГ. - креира се порука  
 fID, име рес., лог. час.<sup>13</sup>, што свима у груни,  
 чека да одобрење свих за улазак у KC;  
~~односно~~ сличан је multicast комутацији  
=> Захтева  $2(n-1)$  првих порука за  
 извиђачу KC; уколико један процес откаже  
 АЛГ. вине не фунционише; може се  
 препрограмити на  $n/2$ , али онда не најдемо  
 потврду; док не добијемо инф. ~~да~~ је  
 предности процес запретно;

Жетон/токен: је сви ресурсе  
 који се креће кроз хостове у круг  
 (ово је најчешће само логика топологија),  
 физички је најчешће магистрала - Ethernet;  
 док процес/хост држи токен, има право  
 уласка у KC; након откупнине токена,  
 процес мора ~~да~~ <sup>потом</sup> ишаћи на свој ред је  
 не гарантује FIFO!; ако се жетон изгуби,  
 проблема, потребна је ресонfigурација

Логоритми избора координатора (leader election)  
 1) **Bully** - Алгоритам - Сви чланови гласају  
 Процесуална са детним бројем, отај који који  
 добије највиши програмира себе координатор  
 2) **Ring алгоритам** - Када се уочи отај  
 координатор, генерички се токен који  
 врзувач мрежовима у логикој топологији преноси  
 (тако да се увећи шанса да ће се додати )  
 Токен током спушта до првога  $i$ , односно  
 определује свој ID као токен и шансу да  
 бидеје; када токен заузети у процесу  
 даји га је генерирао, тај процес програмира  
 новог координатора на основу следећег  
 токена (неколико ID);

~~Разлоги и проблеми репликације, консистентност и постулације~~

**Дисперсивни системи-теорија = Консистентност & Правил. 1 [P6]**

Разлоги репликације: (3)

- Повећање поузданости - даје љубав репликацији  
 ако је наручила, само је подешто компутерске  
 на другу
- Повећање перформанси - распоредљивље  
 оператења (load balancing) међу компјутерима
- Репликација и касиративни су опште прихватљиве  
 технике склопања

Проблеми који уводи репликација - објектне  
 консистентности који може умети дополнити  
 отворење у систему, што се не довољно  
 мешавине може негативно објекту да  
 оно био перформансе

који су консистентне, чак ико read one-  
 резултату најбољи дојом вредна инијијални

~~обратите внимание~~  
! Напомена: Сви ниски кохизионнији су  
ФАТА-СЕНТИВИС

\* Модела: "Нам са је оштреће" - проблем склопимоћи  
које се фронтално уважавају релинса и венчуртија  
да да би се популаризија на глобалном тржишту одвиве  
кохизионним, подједнако је чекавити довољно време на  
покету синхронизацију, што заузимају довољно до  
прада перформанси = због тога је потребно пронаћи  
излатну СРДЧИНУ"

\* кохизионција - чврак се разматра у контексту R/W  
операција; W операција морају да се пропадају  
које се државати кохизионције - време пропа-  
дације и пеноспојаче глобалног члановитика

\* модели кохизионције: (1)

1) структурна 3) клаузулна 1) структурна 3) клаузулна  
2) сесионџулна 4) FIFO 2) сесионџулна 4) FIFO

\* структурна кохизионција - било која R налази под X  
форина резултат последње W оп. налазијом X  
супермнитно чуваја последње глобалне премештаје  
тако да је памоћни појаснији је у оквиру AC-а  
да:  $W(X)_q$  |  $R(X)_q$   $W(X)_q$  |  $R(X)_q$

P2:  $R(X)_q$  | P2:  $R(X)_q$  |  $R(X)_q$  |  $R(X)_q$

очигледно кохизионцијо чврк структурно кохизионцијо

\* сесионџулна кохизионција - резултат било који  
извршенији је исти као да су  $(R \text{ и } W)_q$  свих  
процеса на складишту подложака извршенији у  
неком сесионџулном редоследу и операције  
сваког процеса појављују се у оном редосле-  
ду који је одређен именовим програмом  
> било који премештај R/W је валидно,  
или сви предвиђени види исти премештаје.

\* клаузулна (условна) кохизионција - чврк који се  
погодију ико чланови се стварају јављају у неком  
редоследу; контуранти W се могу видети у  
разију редоследима

- **FIFO консистентност** - У операције једног процеса се види да се друге другим процесом у следећу чврст су издали
- **Нималогичност**
- Систем - немогуће имати у контексту AC
- **Локална сејеренцијална** - потпуно уређена локална
- **Конзистентност** - векторски часовници
- **FIFO** - један број он. чврсто је да процеси појединачно
- **Локална консистентност (Entry Consistency?)** - процес обављају приједу KC, па тај објект се туђује редом највећим приступом свим осетима
- **Модел саје** LOTS. одважарује сопствене групе опреме, а не најчешћи видају се као RW circuit-centric модел консистентности, тобије реплике, дескриптивна табулатура, минималне табулатура, минималне реплике

## Дескриптивни систем-рејса - консистентност & Репл. 2

- **Circuit-Centric** модел консистентности - са са новинама појединачног евиденцијског процеса; клијент је мобилан, те може да чита податке из различитих реплика  $\Rightarrow$  гледа се да које промене може видети клијент саја промени у моделу Circuit-Centric LOTS: (4)

1) Монотони читања (monotonic reads)

2) Монотони уписи (monotonic writes)

3) Читај своје уписе (read your writes)

4) Уписи након читања (writes after reads)

• **Монотони читања** - Ако  $x$  је први читао податак  $X$  на локацији  $L_1$ , свако ново читање тог податка ће вредност имати истију вредност, без обзира на локацију читања

• **Монотони уписи** - Ако процес обави монотони упис податка  $X$  на локацији  $L_1$ , након чега обави иницијалну податак  $X$  на  $L_2$  ако је то се мора.

\* читaj своје чупис - ефект write операција који обављај процес  $\Phi$  нај пољском  $X$  и не вистински ће бити. Видљиво тада при читанju пољског од стране истог процеса  
чупис следи читанje - write операција коју је обављај процес  $\Phi$  нај пољском  $X$  најави пре пољног читанja тог пољског, обавља се нај низом или новијом предношћу пољског  $X$   
\* типови реплика: (3)

1) ПЕРМАНЕНТЕ РЕПЛИКА 2) SERVER-INITIATED 3) CLIENT-INITIATED  
\* ПЕРМАНЕНТЕ РЕПЛИКА - почетни ступ реплика  
реплике иницијираје  $\Theta$  стране објавера - формирају се на иницијативу власника складишта како би се побољшале перформансе и распределио аутогер уском то у неком тренутку долази величи број захтева са неке чланове хостаца; то су привремене реплике постављене у резултату из којих долажу величи број захтева  $\xrightarrow{\text{from files}}$  - стави сервер велики број реплика о броју присуна и остале долазе  
 $\Rightarrow$  АЛГОРИТМ ДИНАМИЧКЕ РЕПЛИКАЦИЈЕ - узима у обзир број присуна неком фјлу и остале долазе тј.  $\text{Cut}(P, F)$ , где је  $F$  фјл а  $P$  сервер; узима  $\text{Cut}(P, F)$  претре  $\text{rep1}(Q, F)$ , фјл се јевном цира на сервер  $P$  и када  $\text{Cut}(P, F)$  падне испод  $\text{del}(P, F)$ , фјл се бризе са сервера  $P$   
 $\Rightarrow$  Сервером иницијиране конјује су најчешће read-only и једине које се могу менјати су перманентне реплике иницијирани стране складиште - познате и као конјути кел; привремена конја су памте конјути да би сматрао време присуна пољашума; могу да јакши проблем континентије, пошто је управљају њим попутно остављено конјути

12

### → Алигаторија Аћурирала: (3)

- 1) ОБАВЕШТЕЊЕ О ОБАВЕШТНОМ Аћуриралу - ИНВАЛИД  
ЦИЈА ОСТАНОВАНИјА КОПИЈА
- ПРЕДНОСТ јЕ НАЧИН СЛОБОДНОГ КРОЗ МРЕЖУ; ДОБР  
РАД ЧИД јЕ БРОЈ Аћурирала и НОГО ВЕЋИ
- ОД БРОЈА ЧИТАЊА
- 2) ПОДАЦИ КОЈИ СУ Аћурирала - САМ ПОДАЦ СЕ  
ПРЕДНОСЕ РЕПЛИКАМА; ДОБР ДЕ КИД ЈЕ БРОЈ  
ЧИТАЊА ИНОГО ВЕЋИ ОД БРОЈА ЧУЛСА
- 3) ОПЕРАЦИЈЕ КОЈЕ СУ ИЗЛУЧАЕ Аћурирале -
  - НАЗИВА СЕ ЈОШ Активно реплицирање<sup>13</sup> | КОМПРО  
МИС ИЗМЕЂУ 1 и 2, ТАКИЧИ МУРДИЧИ СЛОБОД  
РАД ТАКИ ЗАХТЕВА ВИНЕ ПРОЦЕСОР СА ОГ ВРЕМЕНА  
ДЕЛАНИЦА
  - ВЕЋ ОБАВЉАНА ОСВЕЂЕЊЕ ОПЕРАЦИЈА ЧУЛСА
  - КО ИНИЦИЈИРА Аћурирале: (2)

- 1) Сервер (push присул) - Аћурирале СЕ ПРЕДНОС  
РЕПЛИКАМА НА ИНИЦИЈАТИВУ СЕРВЕРА, БЕЗ ДА СУ  
РЕПЛИКА ТО ЗАХТЕВАЊЕ; ОВО СЕ КОРИСИ ИЗМЕЂУ  
ПЕРИМАНЕНТИХ И СЕРВЕРСКИ-ИНИЦИЈАТИВНИХ РЕПЛИКА,  
ОДНОСНО КДА ЈЕ ПОТРЕБНО ОДРВАТИ ТВОСК ЧИВО  
КОНЗИСТЕНЦИЈЕ

- 2) Клијент (pull присул) - СЕРВЕР ИЛИ КЛИЈЕНТ ЗА  
ХТЕВАЊУ <sup>14</sup> А СЕРВЕРА ДА ИМ ПРОСЛЕДИ Аћурирале  
АДА ЈЕ ПОСТОЈАНО; ОВО СЕ ПРИМЕЂУЈЕ КОД Web  
РЕПЛИКА

15

### → Протоколи конзистенције: (3)

- 1) Протокол БАЗИРАНИ НА ПОСТОЈАЊУ ПРИМАРНЕ КОПИЈЕ
  - 2) Протоколи СА ВИНЕ РАВНОПРАВНИХ РЕПЛИКА
  - 3) КАД КОДЕРЕНИИМ Протокол
- АДА СЕ ЗАХТЕВА НАЈВИШИ ТВОСК КОНЗИСТЕНЦИЈЕ,  
ОДНОСНО ОСВЕЂЕЊАНИА, НАЈЧЕШЋЕ СЕ КОРИСИ  
Протокол БАЗИРАНИ НА ПОСТОЈАЊУ ПРИМАРНЕ КОПИЈЕ

glez own protocols between nodes X и Y can  
be used to make sure that a distributed process  
is guaranteed to coordinate between X и Y  
such as Liveness problem no synchronization  
is needed for this purpose (2)

1) remote write protocols - discards  
2) local-write protocols - example of liveness  
locality makes process (e.g. unicast write  
to remote-write protocols

⇒ Hypothesis is that ~~that~~ CE all operations  
are local to a client in a server, primary;  
same copy is stored same as backup

⇒ into distributed system by the nodes of the network  
example - backup = providing read to another  
processor's copy, but CE all write operations  
through memory bus to primary copy

⇒ ~~CE~~ <sup>exists</sup> if <sup>18</sup> implementation of the write operation  
to local-write - process performs primary copy  
+ there is "primary" и <sup>19</sup> consistency locality -  
primary copy is not affected by processes (copy)  
but <sup>19</sup> CE write operation  
through shared write protocols: (2)

1) transfer function 2) establish the consistency  
between functions: <sup>20</sup> - program of the write operation  
is transferred to other function and performs copy. ~~but~~ OB-  
ject; most often implemented via logico-physical  
multicast operation - for each node adds to primary  
consistency

Protocols for consistency:

$$1) N_k + N_w > N \quad 2) N_w > N/2$$

Properties of consistency; maximum consistency

→ Помоћ, логика, отвари, вакуум, преносача, једи  
важнији, али, кире и др. преносачи консисте, посредник од "тешког"  
Данас је РС

## Дистрибуирани системи - теорија - Fault tolerance [08]

1. **Неколико дефиниција** система на дефекте је способност  
система да наступи са исправним функционисањем  
у случају појаве грешака.

- **дефект** је трајни или привремени недостатак
- **логиј** је јављајући недостатак компоненти
- **преносач** је манифестија дефекта и предизвик
- **одуступање** тврдности податка од окоузавање
- **отказ** наступа када систем винче не може  
да обавља функцију за коју је пројектован

### Категоризација грешака:

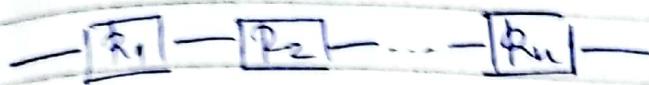
⇒ у односу на трајање: (3)

- 1) пролазне (транзитентне) - исчезао тимеут за  
сличе појаве, или при поновном сличу је  
поправљено послата
  - 2) периодичне - грешака у прекидном стању,  
због постоји неки прекид
  - 3) трајне - грешака у хард-диску, прогрео hw...
- ⇒ друга категорисања грешака:
- 1) "мирне" грешаке - компоненти преостају са  
радом, не генеришу никакав излаз, или  
генерише поруку о грешаку
  - 2) "живе" грешаке - компоненти наступавају  
са радом и генерише погрешне резултате
- **дундуктивна** - општи приказ у пројектовану  
систему отпорних <sup>8</sup> на грешаке
- \* живији редунданс:

- 1) информациони - реализације/кордирате података
- 2) временска - понављају спроводија
- 3) физичка - hw (RAID, бекап), sw (репликација)

\* **неколико дефиниција** систем може толерисати  
грешака, а да наступи са исправним функцијама.

\*Мониторинг параллельных процессов (надежность)  
 які є в компонентах яких є в системі  
 \*Візантійський Протокол, підтримує  
 які компоненти яких є в одній мережі  
 (есе 2012 що є уявлення про мережу, які  
 можуть бути використані для підтримки є ніж  
 \*TMR - Triple Modular Redundancy - використовує  
 як приклад компоненти які є в одній мережі  
 які є в корпоративній інфраструктурі проприєтет



$$R = R_1 \cdot R_2 \cdot \dots \cdot R_n = \prod_{i=1}^n R_i$$

(найменше єдине  
не з порушення)



$$R = 1 - (1 - R_i)^3$$

→ ймовірність

$$R = 1 - \prod_{i=1}^n (1 - R_i)$$

\*TMR є принципом як безпека  
 \*ABFT - Algorithm Based Fault Tolerance - функція яка  
 принципом є обертанням смс та розподілами (результат  
 складається з 3 копій які зберігають...)

\*Erasure coding - складається з k податків або купюра  
 складається з n, де n > k, також якщо є на основі  
 FOF то складається складається з k, керованістю та  
 може використовувати податки складається з (n, k) FOF

\*MDS є Tolerance k-k протилежна

\*Реплікація - найпримітивніша softverова метода  
 як проектувати FT систему

\*Реплікація проблеми можуть бути вирішенні:

- 1) у розподілі пропускні здатності (flat) - всі рівноправні
- 2) у хієрархічному пропускні здатності - поспільні координатори

\*Реплікація може бути:

- 1) пасивна
- 2) активна

Завдання єз Протокола/організації

Владивате реализација = једнодимензиона за ИМЛ, али  
који функционални у СУМЦУ постоји толерантан  
јеско превале (шта ће тумар Грешак?)

Чо је дати начин за посматране толерантноста <sup>16</sup>

Визуелноје Грешак је кроз активну реализацију  
тусагаштавате <sup>17</sup> (консензус) при постојану <sup>18</sup> визу-  
елских премика - избор координатора, приступ к  
његовим протокол који може да предаваје

ненужане конфиденцијалне информације

\* BGP (Byzantine General Problem) - ненужане

примесори, поузданите који имају

\* Установио имамо ши ненужане, потребно <sup>19</sup>  
да је најмање  $2m+1$  поузданих (ук.  $3m+1$ )

⇒ Потребно је обавити шији разнице разлике  
порука (свако свакоме) да се постапа као

⇒ Крло неправдично решење њу осврту PC, због

веомиког оптерећења преце њији масац  $-O(mn^2)$  нп  
( $m$ -надимач,  $n$ -чекуље) ⇒  $n = 3m+1$  <sup>20</sup>

\* Сепаратије за опоравак: (2)

1) backward recovery - пратње Checkpoint-a

и вратије систем у претходно стање

2) forward recovery - преводије систем у

Трећијија, погрешнији, и ново, исправито, стање

\* Алијаја опоравка - конвјенцијији преце:

$(e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C$

<sup>21</sup> малијади до великаје

\* Commit Effect - сваки процес бележи свог

Checkpoint-а због синхронизације са осталим

⇒ да је ово спремно <sup>22</sup> кориси се

2-phase commit checkpoint - координатор

што свима REQUEST, када сви одговоре са  
ACK, координатор потврђује са DONE

\* Клиент које може наступити под RPC-A: (1)

1) Клиент које у стапу да ~~помира~~ <sup>помира</sup> срвбер

2) Захтев клиента ка срвберу је иктулан

3) Отказ срвбера иако преузима захтева од клиента

4) Одговор срвбера клијанту изгубљен  
адреса, шта прати, података, Statefulness, пребачивоста податка

## Дистрибујати систем-теорија-ФАС [ФАС]

\* ФАС <sup>1</sup> иметован Скуп података тражено залишених

\* дистрибују - специјална врста фјла која се користи за организовање других фјлова и

хиперхвјасе <sup>3</sup> структуре (сабло мапинг)

\* ФАС систем- компоненти определитељног система која служи за организацију, складиштење, иметовање, претраживање, анонсе и заштиту фјлова; гарантије, обезбедује и интерфејс за рад са фјловима именовање фјлова <sup>4</sup> на диску: (3)

1) континуално 2) улочитно 3) итерактивно <sup>5</sup>

\* дистрибујати фјл систем - све компоненте регуларног фјаја система уз компоненте за клијент

- срвбер комуникацију, спроводи се од 1) (2)

1) дистрибујати сервис Срвиста - иметовање и локирати пресликавање текстуалног имена фјла у јединствени фјл идентификатор (на нивоу чланог AC); може бити

~~дес~~ на засебној, одвојеној мапи

2) фјл срвист - обрачавање садржат фјла;

напаки се локално, на сабло мапи <sup>6</sup>: (6)

\* ШТА ДФС ТРЕБА <sup>7</sup> да обезбеди: (6)

1) транспарентност 4) транспарентност 1) присутица 4) транспарентност  
приступа 2) локално на 5) миграционе 2) присутица 5) миграционе  
транспарентност 3) транспарентност 6) хиперредакције 3) присутица 6) хиперредакције

2) локално на 5) транспарентност 2) локално на 5) транспарентност  
транспарентност 3) транспарентност 6) хиперредакције 3) транспарентност 6) хиперредакције

3) транспарентност 6) хиперредакције 3) транспарентност 6) хиперредакције

7.

Узроковац приступа удаљеном фајлу:

- 1) upload/download податак - једини сервиси су read и write - преузима се фајл, неће нико што и чуписује назад на сервер је перформансе су добре због што нема преватног сабратаја
- 2) Такође немајте фајла је проблем:

- шта да клијент нема простор да започне нови фајл; шта да клијенту дозволи само 100 фајла; шта да други <sup>пјавити</sup> испостављенији корисници не модификују исак фајл

- 2) подел удаљеног приступа (remote access)
  - обезбеђено је извршење стапајућих онај редијул над фајловима (open/close, read/write by user, ...)
  - на удаљеној машини, тј. на серверу које се фајл отворио налази; ~~даје~~ ~~даје~~ имена проблеме претходног подела, тије због додатне компликсности коју нори да обезбеди
  - има касније перформансе и нове <sup>даје</sup> <sup>који</sup> <sup>се</sup> <sup>се</sup> затушена мрежа (<sup>предња</sup> <sup>који</sup> <sup>се</sup> <sup>се</sup> <sup>погоди</sup> <sup>тако</sup> <sup>чека</sup>) приступу

на серверу могу бити:

- 1) stateless сервер - не памти никакве информације о клијенту нити о приступима нају фајловима, отпорнији је на отказ сервера чудаво због недостатка информација - нема шта да се изгуби, клијент паки све
- 2) stateful сервер - сервер памти све информације везане за клијенте, приступе фајловима, и даје <sup>10</sup> ако је отворити викендацију, али тек посматра <sup>11</sup> мора се перформансе правити check-point

у пројектантске питања: (3)

- 1) именовање (некупале им се сопствен члан)
- 2) сем антреја (дуготрајнији фајлови)
- 3) подација база помоћи учица

Именование файла - Имя файла должно быть такое, чтобы открыть конкретную локацию файла (локальная проприетарность) и для будущего использования файла не меняло для прочтения локацию (локальность неизменности), т.е. неприватность (локальность неподеленность именования).

1) комбинация host:file\_path

2) идентификатор 3) ТОЧКАННАЯ ИНТЕРПРЕТАЦИЯ (дистанционный просмотр)

Семантика имени файла:

1) ОСВЕЩЕНИЕ ИМЕНИ СЕМАНТИКА (Unix) - реальный сервер, без кеширования на стороне клиента, так как сервер вычисляет, какие перформансы

2) => может обновлять кеширование на стороне клиента с Write-through политикой записи

ибо это побоюване перформансов

2) Семантика сессии - Сервис прослушивает файлы между открытием и закрытием файла, проверяя, что видимые на них закрытия файла (основано на конвейере 15 открытия открытия другого процесса)

Кеширование - в память сервера или на сторону клиента

каким же образом кеш?

=> та же кешировка списка (3) 16)

1) write-through 2) запись 3) запись + промежуточные файлы

=> А мы с чьими же усилиями кешится кешиться се мастером?

1) push прослушивание 2) pull прослушивание

Использование систем-теорий - GFS & HDFS Г10.3

А мониторинг - образца всемирной комиссии по ядерной безопасности (комиссия которая проводила все международные ядерные инциденты)

→ Scale-Out <sup>2</sup> >> Scale-Up - по перформанси и CPU

Scaling-out имплементира дисподибулутн систем

→ MapReduce - најчешћи обраћејеј јављају за Big Data

→ GFS (Google File System) + MapReduce VS

Apache HDFS + MapReduce - систем за

складиштење и атакиву врлике компоненте података

→ Hadoop = HDFS + MR - ~~односно~~ Big Data систем

→ Node <sup>4</sup> - рачунар који представљају симосити

(чубичаји) hw који садржи податке

→ Rack - колекција више извора који су физи-

чески близу један другом и повезани истим

мрежним свицем (јасно тачно!)

→ Hadoop cluster <sup>6</sup> - колекција ресурса

→ Блокови <sup>7</sup> - величине 64MB (128MB за новије)

- најчешћа јединица података којом чувају

HDFS - сваки датац се дели на одређен број

блокова, сваки блок се реплицира (дакле рен-

же no default - 3) по принципу at most 1

per node, at most 2 per rack <sup>8</sup>

=> ОВИДЕЋЕ РЕБУЈАНО - предвиђено је <sup>9</sup> број блокова

по датују његов диске врлике, фајл може

имати већи број блокова који ће бити дисперсивнији

FAT његов репликација

=> Hadoop репл. се може изменити и чак посмат-  
рати за сваки подгрупацији датац

→ MapReduce компонента - дате вредности трансформи-

шују - Map и Reduce - извршена обе вредности се

извршавају у паралелно

→ вредности компоненте у Hadoop кластеру - кластер

извор, master извор и slave извори

→ кластер извор - load-ује податке у кластеру,

испраћа MapReduce job и ботије има преведе-

нећујући извор извршача

Master чвртју се врши највећије складиште и извршитеља. Једног чврта у оквиру кластера а славе чвртеви обично су извршници и складиште за то да се они су заправо отворију.  
Hadoop daemon-и: (2)

- 1) HDFS daemon-и 2) MapReduce daemon-и
- \* у HDFS daemon-у опадају: (2)
  - 1) NameNode (NN) daemon 2) DataNode (DN) d.
  - => посебан симбол је ~~да~~ NN и други DN - NN је вредна координатора, а DN служи за управљање података и давање информација NN-у
  - \* NN одржава физички простор и сваки промета података користије је зато што оно је оптимизовано
  - => NN користи инф. о томе како су физички податаки на блокове, који садрже чвртеву чврту које блокове и сам чвртев је пресликати
  - => блокови су вредноста физичкији
  - => за комуникацију са DN-овима, NN користи heartbeat <sup>12</sup> највеће - DN имаје обавезу да дати NN-у сваке 3sec, а сваки том објекту је сваки сајт IP-адреза блокова које садржи; NN ће стражи о томе да сваки блок је довољан факторјејтаже
  - => NN ће највеће извршити највећи јавни сервис, који је представљен master сервисом, да је једна приступачна тачка буџету за кориснике <sup>13</sup>
  - => NN ће податке обезбедити за организацију и праворобство на DFS-у чврта у глашној меморији:
  - \* 1) file namespace 2) file to block mapping 3) block locs
  - \* тројство чвртанаје метаподатака - NN ће чвртју у локацијама фс-у, TBS - Checkpoint-овима, и освежији fsimage <sup>14</sup>
  - \* фалт је кога је тројство извршило промене које се edit log <sup>20</sup>

1. file sys, namespace
2. мониторинг файлов на блокове
3. локализация блоков

⇒ ДАКЛЕ Ако је мешавина у ГЛАВНОЈ МОДУЛУ  
 СЕ ОБЕМОДУЛИЧИМ СНЯПШОТ-ОВИМ који се извршију  
 у оквиру ~~edit log-a~~<sup>21</sup>; ОВЕ ИЗМЕНЕ У ОКВИРУ  
 HDFS-а симулација ~~edit log-a~~<sup>22</sup> и edit log-у

~~edit log~~ ~~edit log~~<sup>23</sup> не је занад  
~~snapshot~~ ~~snapshot~~<sup>24 ово први пут  
~~занад~~ ~~занад~~ се ово уметају snapshot-у, edit log-у  
 ⇒ edit log се применjuje на ~~fsimage~~ јер то  
 приликом restart-a NN-а, на употребу превише  
 тешко, restart HDFS-а; занад HDFS-а  
 чио велике формирати CP-а да би се преузима-  
 ћуло даји проблем корисци се обустављају NN,  
 који је посао јесу да конкурира са <sup>25</sup> при-  
 напади NN-а, применjuje edit log на ~~fsimage~~  
 (не опредељена функција за откас <sup>26</sup> се примарниот NN)</sup>

⇒ Rack awareness - NN због физичку локацију  
 DN-ова, да би могао блокове да <sup>27</sup> распореди  
 по приступу <sup>28</sup> 2 per rack, 1 per node  
 ⇒ HDFS уник-кластер разбија фајл на блокове,  
 према A,B,C, j контексту NN и за сваки rack  
 добија месту од 3 DN-а (дакле реч је 3)  
 - за ово NN користи свог rack awareness; пре  
 униса, кластер HDFS потврди да су ови DN-ови  
 определјени да прими блок - тада добије потврђујући  
 креће уник; DN-ови при уник трајују резултате  
 и редоследу који имплементира распореде од нек-  
 формија

⇒ Ако NN преиграе да организује heartbeat-ови  
 да користи DN-а, ~~не~~-реплицирају блокове који су  
 се укапају, користи rack aware податак  
~~помагајте~~ из HDFS-а <sup>29</sup>-контекстујући CE NN да користи  
 ово DN-ови да организују блокове организујући и буџет  
 тражења и аукције (према)

## \* GFS vs HDFS: (±)

- 1) Master ≈ NameNode 2) Chunkserver ≈ DataNode
- 3) operation log ≈ (edit log) 4) chunk ≈ block
- 5) random writes vs. only append
- 6) multiple wr., multiple re. vs single wr., multiple re

## Дистрибутивни системи - Теория - Общие темы [10-15]

### I ЧВод [Ø1]

- 1) појава DC - чврди, разложи 2) Дефиниција DC
- 3) предностим DC над централизованим системом
- 4) мате DC у односу на ~~андр~~ централизован систем
- 5) основне особине DC и поделачица објекта
- 6) middleware слој и подела имп. протокола
- 7) подела дистрибутивних система / 7

### компјутерни системи I

### II RPC - Remote Procedure Call [Ø2]

- 1) проблеми код RPC - A 2) пренос параметара
- 3) формат при позиву RPC - A, client и server & stub
- 4) логиките удаљеног сервера - статично и динамично
- 5) семантика трагетаса код RPC - A
- 6) Sun RPC - граден филови, XDR дефиниција, клијентски позив серверске процедуре, писање RPC код - A за Sun RPC систем (клијент и сервер са port mapper (rpcbind на 111) и повезивање (binding), предност Sun RPC система)
- 7) DCE RPC - DCE сервис, RPC сервис, дистрибутивни сервис, безбедносни сервис, дистр. фул опсервис дистрибутивног времена, писање DCE RPC програма (IPX, семантика позива, клијент, сервер, иницијализација клијента са сервером) / 7

### III Комуникација II [43]

и садржавј

- 1) RMI - Remote Method Invocation
- 2) Remote Interface, proxy, skeleton, remote object reference
- 3) Java RMI 4) Повезивање клијента и сервера - binder
- 5) Кораци у повезивању клијента и сервера (G)
- 6) пренос параметара код RMI +
- 7) типови комуникација - критеријум поделе 14
- 8) Перзистентне и транзитивне 9) Синхронне и асинхронне
- 10) Адисcretне и стримовске 11) Нагнутије комуникације (G)
- 12) MOM - Message Oriented Middleware - дефиниција и сервер
- 13) JMS представи и типови код 14) MQS - Message Queue bus.
- 15) Чубичајени интерфејс код MQS + (4 опреџије)
- 16) Point-to-point принцип комуникације, различите конфигурације
- 17) Архитектура система са редовним порукама - QM, рутер
- 18) Pub/Sub модел 19) publisher, subscriber, topic, filter
- 20) Pub/Sub интерфејс 21) филтрирање по теми и садржају
- 22) JMS - Java Messaging Service 23) JMS клијент, JMS производјер JMS producer, JMS consumer, JMS topic, JMS queue
- 24) Административни објекти - фабрике конекција и редови/теше
- 25) програмирање са JMS API, ток креирања објекта
- 26) систем и типови JMS порука 27) потрошачи и производачи
- 28) JMS P2P класе 29) JMS Pub/Sub класе (но G)

129

### IV Синхронизација 1 [84]

- 1) Синхронизација - појам и врсте 2) физички часовници
- 3) Кристални алгоритм (анх. физички часовници)
- проблеми и предности 4) Berkeley RTC
- 5) NTP - Network Time Protocol 6) ревизији NTP-A (3)

16

### V Синхронизација 2 [85]

- 1) логички часовници - који проблем решавају

реалитивно уређење

## II Репликација најраније before

- 2) реплиција "десно се пре" ( $A \rightarrow B$ ) и осовине, ако има
  - 3) временске маркиње као пр. чла и логоритам
  - 3a) заузетосте before реплиције око којима
  - 4) проблем стремелских маркиња - недостатак информација
  - 5) потпуно употреба групних консистентности базирата на  $B$ .
  - 6) векторски чланови - решење проблема, ~~неконсистентна векторска маркија~~
  - 7) нови алгоритам и кореције вектора ~~који информацију~~
  - 8) чување исказивања и ~~антиалгоритам~~ алгоритам (3)
  - 9) имплементација алгоритам ~~добра стране, лоше стране~~
  - 10) дистрибуирани алгоритам (Ricart & Agrawala), принципи и могуће измене 11) Ring алгоритам, осовине (4)
  - 12) алгоритам избор + (election alg.), чла и осовине
  - 13) Bully election алг. 14) Ring алгоритам репликација  
консистентност
- /14

- ## VI Консистентност репликација [Ø6]
- репликацијом
- 1) разлоги репликације (3)
  - 2) проблем чувања ~~не пуштај да се прегледа, а битно је~~
  - 3) модели консистентности ~~дана~~ ~~центри~~ ~~због конфлікта~~
  - 4) структура консистентности, дефиниција, проблем
  - 5) каснијија консистентности; дефиниција; примери
  - 6) условна/каузална консистентност; дефиниција је пример
  - 7) FIFO консистентноста дефиниција, кореције он десните
  - 8) слаба консистентност - разлике у односу на осовине
- /8

- ## VII Консистентност репликација [Ø7]
- 1) Client-centric модел консистентности - процесна перспектива
  - 2) монотони читања (monotonic reads) - деф. и пример
  - 3) монотони уписи (monotonic writes) - деф. и пример
  - 4) читај своје уписе (read your writes) - деф. и пример
  - 5) упис следи читање (writes follow reads) - деф. и пример
  - 6) реплицираје позиционирање реплика
  - 7) перманентне реплике
  - 8) реплике иницијираје од стране сервера
  - 9) АЛГ. динамичке реплике
  - 10) реплике напомијавају о спаљењу

- 11) Асистентија - твориране - подаци, логији, оптерачи
  - 12) иницијативи твориране - симвер (push), конјан (pull)
  - 13) протоколи коинциденције и подела (3)
  - 14) протоколи застановата на примарата копија (2)
  - 15) remote-write протоколи - 2 верзије и СЕКВ. коад.
  - 16) local-write протоколи 14) реплицирали write протоколи застановата на кворум и активна репл.
  - 17) Гифордова идентичност за кворум протоколе
- /18

## VIII Fault tolerance [08]

- 1) Адекват, тренше, откази - основни појмови
  - 2) Категоризација грешака - у односу на Тријаде и на начин "функцијотивна" (подела)
  - 3) редундантна и врсте (3)  $\xrightarrow{\text{неда отпорност}} k$ -тOLERАНТНОСТ
  - 4) ~~5) k-тolerантност за ширте и визуелнице тренше~~
  - 5) физичка редундантна - TMR (Triple Modular Red.)
  - 6) ABFT и примене 8) Erasure coding - принцип,  $(n, k)$
  - 7) репликација како мера против FT
  - 8) организација процеса - flat и хиерархија
  - 9) активна VS. Активна репл. из угла FT -  $\xrightarrow{\text{активна репл.}}$
  - 10) коинциденција у присуству грешака - као решение
  - 11) BGP - Byzantine General Problem - како често се користи је потребно да има и издајачи? Како се разделя порука? (зашто и како)
  - 12) BGP пример за  $n=1 \quad n=4$  (чому ишто  $n \geq 3$  и да?)
  - 13) Forward и backward recovery (Checkpointing)  $\xrightarrow{\text{преди}}$
  - 14) Checkpointing - да доведе до ефикаснији коинциденцији
  - 15) координисано бешење стапа - Z-Way, Commit
  - 16) тренше које могу настапит код РРС -
  - 17) објавитељ и реплена
- /18

## IX DFS Гаг

- 1) файл, директоријум, фјл систем-основни појмови
- 2) локитни смештатија фјловат ~~на диск~~ (3) <sup>+ идентификатор</sup>
- 3) дистрибуирани фјл систем - <sup>основне</sup> особине и компоненте
- 4) шта треба да обезбеди ДФС (6)
- 5) модел приступа удаљеном фјлу (2)
- 6) upload/download модел - локитни функционализам <sup>предност</sup>
- 7) модел удаљеног приступа (remote access) - -||-
- 8) Stateful и Stateless ~~сервери~~ - упоредување на вртеска са кохизиције и расподелције
- 9) проектиранска питања (3) - именовање, семантика десетија фјловат, потчија сеја
- 10) именовање фјловат - локална и мрежна пратљареност, 3 приступа
- 11) именовање компонентијам имена ~~host~~ и лок. имена <sup>фјл</sup>
- 12) mounting 13) тотално интегрисано именовање
- 14) СЕВЕРНИЧАНА СЕМАНТИКА десетија фјловат
- 15) СЕМАНТИКА СЕСИЈЕ - десетије фјловат
- 16) кеширање - меморија сервера VS. <sup>каширање на</sup> апаратни куфери
- 17) политика чука - write through, late write, on-close
- 18) push и pull ~~иницијација~~ #втурити
- 19

## X GFS & HDFS

- 1) scaling-up vs. scaling-out 2) Hadoop-DFS+MR
- 3) компоненте Hadoop-A - DFS и MapReduce
- 4) Node, Rack, Cluster - основни појмови информасије
- 5) Hadoop блокови - принцип, основна јединица/веничина
- 6) фактор репл. блокова - default 3
- 7) Врсте извора - клијент, master, slave и тихове чуке 8) Hadoop десмонт - MR и HDFS десмонт
- 9) HDFS десмонт - NameNode и DataNode

- 10) NameNode - функција, методација која садржи  
информације о поседуји и ~~имају~~ ком физичку гравитацију  
предавање ~~дана~~<sup>1b-a</sup> блока у локални, фасада  
11) heartbeat месец ; 3sec / сваке 1000ms  
12) Трајно очување методација ~~fsimage и edit~~  
13) NN репарти и дефрагментација ~~NN~~  
14) Rack awareness ~~NN~~- ~~at most 1 per node,~~  
~~at most 2 per rack~~  
15) HDFS уник-филе профес 16) HDFS репрезентација  
17) HDFS штапе-процес 18) конзистентност у GFS  
19) употребљавање HDFS-а и GFS-а  
20) конкурентни write 21) разлика HDFS и GFS