**Уточненный псевдокод основных функций p2p алгоритма Chord ENECUUM**

ОБОЗНАЧЕНИЯ:

n – адрес текущего узла. Состоит из полей: ip и id

ip – IP адрес узла

id – 160-битный идентификатор узла в сети Хорда: id = sha1( ip )

n.send – на текущем узле с адресом n вызывается процедура send для отправки сообщений

confParams – начальные конфигурационные параметры Chord:

* seed //адрес загрузочной ноды(-ы)
* ip //IP адрес текущей ноды
* timeouts //тайминги таймеров
* counters // счетчики повторной отправки
* priorities //приоритеты очередей
* max\_deep //глубина блокирующей обработки очередей

//Процедура отправки сообщения <type> адресату <d>. В параметрах также указывается первоисточник сообщения <i>; от какого непосредственно узла идет отправка <s>; **ID ноды, которую ищем <id>**

n.**send**(type t, initiator i, sender s, destination d, **uint160 id**)

//Та же процедура отправки сообщения. Содержит **флаг <isJoin>, что инициатор пытается подключиться к сети**

n.**send**(type t, initiator i, sender s, destination d, uint160 id, **bool isJoin**)

//Та же процедура отправки сообщения. Содержит **найденный адрес ноды <f>**, включающий ip и id

n.**send**(type t, initiator i, sender s, destination d, **found f**)

//Та же процедура отправки сообщения. Содержит дополнительный параметр адрес предыдущего узла **<p>**

n.**send**(type t, initiator i, sender s, destination d, found f, **predecessor p**)

type = {“join”, “find\_succ”, “succ”, “find\_pred”, “pred”, “notify”}

Каждое сообщение должно содержать счетчик restransmit counter – количество пересылок от узла к узлу.  
Если счетчик достигнет величины 2x logN, то сообщение следует отбросить, не обрабатывать.

**Описание процедур и функций алгоритма Хорды**

// Включили ноду, нода выполняет процедуру create

n.**create**( confParams )

reset

params = confParams

n.id = **sha1**( params.ip ) //n – адрес текущего узла

predecessor := n

successor := n

**for** i = 1 **to** m

finger[i] = n

searchedID = n+1

i = 1

**while** (params.seed != null)

//текущая нода n отправляет сообщение join к загрузочной ноде n`

n->**send**(“join”, n, n, params.seed[i], searchedID )

i = i+1 **mod** params.size

**wait**(reply) //ожидание прихода ответа <reply>

**if** reply **is exist**

successor = reply.found //поле <reply.found> содержит адрес для successor

finger[1] = successor

predecessor = reply.predecessor //поле <reply.predecessor> содержит адрес для predecessor

**plan**(fix\_fingers, period) //запланировать выполнение fix\_fingers по периоду

**end**

**end**

//Нода, получив сообщение <join>, вызывает процедуру c таким же названием join

//Процедура join на вход получает параметры:  
//- initiator – адрес первоисточника сообщения

//- sender – адрес непосредственно кто отправил

//- destination – адрес назначения

**join(**initiator init, sender s, destination d**)**

**if** d.id != n.id //sha1 ноды назначения d не совпадает с sha1 текущей ноды n

**remove** //отбросить сообщение, не обрабатывать, оно не для нас

**return** //принудительно завершить процедуру

**if** params.seed == **null**

params.seed += init

**plan**(fix\_finger, period) //запланировать выполнение fix\_fingers по периоду

sID = init+1 //<sID> расшифровывается, как searchedID, будем искать successor для ноды i

isJoin = true //мы находимся в ситуации, что нода i к нам коннектится

action = **find\_successor**(id, init, s, isJoin, &addr)

**if** action == **REPLY**

n.**send(“**succ”, n, n, init, n, predecessor**)** //отправить первоисточнику <init> ответ <succ>,  
 //включая свой адрес <n> и <predecessor>

predecessor = init

**if** successor = n //узел был один в сети и его successor равнялся n

successor = init //вносим ноду i в наш successor

**else if** action == **FORWARD**

n.**send(“**find\_succ”, init, n, addr, sID, isJoin**)** //пересылаем <join> дальше по сети

//в виде сообщения <find\_successor>  
 // c флагом<isJoin>

**end**

**bool stabilize()**

n.**send(“**find\_pred”`, n, n, successor, successor.id-1**)**

**wait**(reply, timeout) //ответ <reply> должен содержать адрес найденной ноды <found>,  
 //включая поле <id>

success = **false**

**if** (reply **is exist)** **AND** **(**reply.found.id **∈** (n.id; successor.id**)** )

//вызывается функция**isInRangeOverZeroNotInc**

successor = reply.found //здесь поле <found> содержит predecessor для ноды successor

fingers[1] = successor

success = **true**

n.**send(“**notify”, n, n, successor, successor**)**

**return** success

**end**

//Нода, получив сообщение “notify”, вызывает процедуру c таким же названием notify

//Процедура notify на вход получает параметры:  
//- initiator – адрес первоисточника сообщения

**notify(**initiator n’**)**

if (predecessor = **null)** **OR** (predecessor = n) **OR** ( n’**∈** (n.id; successor.id) )

predecessor = n’ //вызывается функция **isInRangeOverZero**

**end**

**fix\_fingers()**

next = next + 1

**if** next > m //m – количество finger в ноде

next = 1

**if** next == 1

**if** n.**stabilize()** == **false** **// выполнить стабилизацию, надо восстановить свое место в кольце, т.е.**

i = 1 **//** установить succ и pred

**do**

finger[next] = n.**send**(“find\_succ”, n, n, params.seed[i], n.id+1) //ставим succ

**wait**(reply, timeout)

**if** reply **is exist**

finger[next] = reply.found

i = i+1 **mod** params.seed.size

**repeat** n.**stabilize**() == **true** //ставим pred

//проще вызвать create без сброса фингеров и очередей, не проверялось

else

n.**send(“**find\_succ”, n, n, successor, n+2next-1)

**wait**(reply, timeout)

**if** reply **is exist**

finger[next] = reply.found

else

n.**stabilize() //выполнить стабилизацию**

alive = n.copyPrevAliveFinger() //записывается последний «живой» фингер – пингующаяся нода

**if** alive **is exist**

finger[next] = reply.found

**else**

**reset** //все фингеры мертвые, перезагрузка Хорды

**create**

**return**

**wait**(period)

**end**

**check\_predecessor()**

**if** predecessor **has failed**

predecessor = n

n.**send(“**find\_pred”`, n, n, successor, n.id-1**)**

**wait**(reply, timeout) //ответ <reply> должен содержать адрес найденной ноды в <found>

**if** reply **is exist**

predecessor = reply.found

**end**

//Функция внутри ноды n осуществляет поиск фингера, у которого id будет близко или равно фингеру

//Функция <find\_successor> на вход получает параметры:

//- id - искомый ID ноды, тот что ищем

//- initiator – адрес первоисточника сообщения сообщения

//- sender – адрес непосредственно кто отправил собщения

//- destination – адрес назначения сообщения

//- isJoin – флаг <isJoin>, что инициатор пытается подключиться к сети

//- found – сюда запишется найденный адрес ноды из фингера

**chord\_action** **find\_successor(**uint160 id, initiator init, sender s, bool isJoin, &found**)**

action = **DO\_REPLY**

found = **null**

//Порядок проверок важен! Проверяем: себя n, затем pred, [pred; n], [n; succ] и наконец fingers

**if** id == n.id

action = **DO\_REPLY**

found = n

**else if** id == predecessor.id **AND** predecessor != **null**

action = **DO\_REPLY**

found = predecessor

**else if** predecessor.id **AND** predecessor != **null**

action = **DO\_REPLY**

found = predecessor

**else if** id **∈** [predecessor.id; n.id] //вызывается **isInRangeOverZero**

action = **DO\_REPLY**

found = n

**else if** id **∈** [n.id; successor.id**]** //вызывается **isInRangeOverZero**

action = **DO\_REPLY**

found = successor

**else**

//Дойдя до сюда, значит, не нашлось ноды в диапазоне от [pred; n] OR [n; succ]

//Проверяем фингеры в надежде найти «впереди» ближайшую ноду к id

//Выполняем цикл closest\_preceding\_node с модификацией

**for** i = m **downto** 1

{

**bool** notInitiator = finger[i] != init

**bool** notSender = finger[i] != s

**if (**finger[i] **∈ [**n.id;id**]) AND (**notInitiator == **true)** **AND (**notSender == **true)**

{ //вызывается **isInRangeOverZero**

found = finger[i]

**if** i-1 >= 0

**if** finger[i].latency **>** finger[i-1].latency //это механизм Low Latency Chord

found = finger[i-1]

**break**

**}**

**}**

**if** found == **null**

**{**

//Фингер не был найден, пытаемся отправить вперед к successor или successor+1 …

//Для этого вызываем <find1stFingerNotForbidden>

found = **find1stFingerNotForbidden(** {init, s, n} **)**

**if** found == **null**

{

//Вперед мы тоже не может отправить, т.к. именно оттуда и пришло сообщение и там

//расположены init, s или у нас фингеры указываются на самих себя n.

//Тогда остается один путь - назад

**if (**predecessor **!=** s) **AND** (predecessor **!=** init) **AND (**predecessor **!=** n)

found = predecessor

action = **DO\_FORWARD**

**else**

//Остается только вернуть собственный адрес <n>

found = n

action = **DO\_REPLY**

**}**

**else**

//Идем вперед по кольцу, т.к. только туда еще можно идти

action = **DO\_FORWARD**

**}**

**else if (**found == n**)**

**{**

//Пытаемся найти фингер такой, что finger[i-1] <= id <= finger[i]

//Такой фингер представляет собой successor для искомого id и превосходит его численно

closerToTarget = **findAlitlleBigFinger(**id, {s, init, id} **)**

**if** found == **null**

action = **DO\_REPLY** //found содержит n, вернем <n>

**else**

found = closerToTarget

action = DO\_FORWARD

**}**

**else**

//Вернем стандартно найденный фингер

action = **DO\_FORWARD**

**if (**isJoin**) AND (**action == DO\_REPLY**) AND (**found != n**)** //Если к нам обратилась нода, которая

action = **DO\_FORWARD** //подключается к сети и мы не являемся её

//successor, то переслать сообщение “find\_succ”  
 //непосредственно successor для этой ноды.

//Это делается, чтобы непорсдественно нода

//successor могла вернуть свой predecessor  
 //и не потребовался отдельный запрос для   
 //получения predecessor

**return** action

**end**

//Среди всех фингеров находим первый фингер, который впереди по кольцу и не содержится в списке запрещенных для рассмотрения фингеров <forbidden>

**address find1stFingerNotForbidden(**list<uint160> forbidden**)**

**for** i = 1 **to** m

**bool** accept = **true**

**for** j = 1 **to** mm //mm – размер forbidden

**if** finger[i] == forbidden[j]

accept = **false**

**break**

**if** accept == true

return finger[i]

result **null**

**end**

//Среди всех фингеров находим первый фингер, который впереди по кольцу, не содержится в списке запрещенных для рассмотрения фингеров <forbidden> и при этом finger[i-1] <= id <= finger[i]

**address findAlittleBigFinger (**uint160 id**,** list<uint160> forbidden**)**

**for** i = 2 **to** m

**bool** accept = **true**

**for** j = 1 **to** mm //mm – размер forbidden

**if** finger[i] == forbidden[j]

accept = **false**

**break**

**if (**accept == true**) AND (**finger[i-1] <= id**) AND (**id <= finger[i]**)**

return finger[i]

result **null**

**end**

**bool isInRangeOverZero(uint160 id, uint160 from, uint160 to)**

**if** from <= to //Generic order, no over zero, [from; to]

**if** id >= from **AND** id <= to

**return true**

**return false**;

**else**

//Over zero,[from -> max; 0 -> to]

if id >= from **AND** id <= MAX\_UINT160 //[from; MAX\_UINT160]

**return true**

if id >= 0 **AND** id <= to //[0; to]

**return true**

**return false**

**end**

**bool** **isInRangeOverZeroNotInc(**uint160 id, uint160 from, uint160 to**)**

**if** from < to //Generic order, no over zero, (from; to)

**if** id > from **AND** id < to

**return true**

**return false**;

**else**

//Over zero, (from -> max; 0 -> to)

if id > from **AND** id <= MAX\_UINT160 //(from; MAX\_UINT160)

**return true**

if id >= 0 **AND** id < to //(0; to)

**return true**

**return false**

**end**