Network Management Project

Jiannan Guo - ICT-Innovation
Niklas Semmler - ICT-Innovation

2. Dezember 2013

1. Task

1.1. Pseudocode

The following pseudo code (listing 1) is inspired by [1]. To the existing object A a new object Comm was introduced to handle all message exchange.

Listing 1: Pseudocode for $\widetilde{f}(t)$ with rate control

```
messages:
(ResponseTimeArriveMessage, responseTime) # similar to local
(UpdateVector, neighbor) # similar to new and update
(TimeOut, responseTime)

objects and help functions:
A       # storing values for and performing aggregation
Comm    # communication messages
A.aggregate_local       # aggregate local response times for local statistics
A.aggregate_sub_tree    # combine local values with children's for statistics
                        #  of the whole subtree
A.value                 # values of statistics of subtree (including own)
Comm.send_to_self       # send a message to the own node with a delay
Comm.send_to_neighbors  # send to all specified neighbors
Comm.send_to_parent     # send to parent
find_parent             # among neighbors, choose first with minimum level as
                        #  new parent
constants:
DELAY := # fixed time window
INF := # really big number

global variables:
msgBudget = 1

procedure GAP( )
    Neighbors := empty;
    ResponseTimes := empty;

    if v = root then
        level = 0
```

```
            parent = 0
        else
            level = INF
            parent = INF
        end if

        A.initiate()
        while true do
            read message;
            switch (message)
                case (ResponseTimeArriveMessage, responseTime)
                    ResponseTimes.add(responseTime)
                    # to keep only a set of responseTimes in cache
                    Comm.send_to_self(DELAY, TimeOut(responseTime))
                    A.aggregate_local(ResponseTimes)
                    A.aggregate_sub_tree(Neighbors)
                case (UpdateVector, neighbor):
                    Neighbors.add(neighbor)
                    (newParent, newLevel) = find_new_parent(Neighbors)
                    A.aggregate_sub_tree(Neighbors)
                    if (level != newLevel || parent != newParent) then
                        level = newLevel
                        parent = newParent
                        Comm.send_to_neighbors(Neighbors, level, parent, A.value())
                        continue
                    end if
                # to keep only a set of responseTimes in cache
                case (TimeOut, responseTime):
                    responses.remove(responseTime)
                    A.aggregate_local_value(ResponseTimes)
                    A.aggregate_sub_tree(Neighbors)
            end switch
            if (msgBudget > 0) then
                Comm.send_to_parent(Neighbors, A.value)
                msgBudget--
            end if
            if (msgBudgetReset) then # waiting for another program to change
                msgBudget = MSG_BUDGET
            endif
        end while
    end procedure
    procedure ResetMsgBudget()
        msgBudget = 1
    end procedure
```

1.2. Implementation Details

A new base class was implemented which all of the tasks extend: `peersim.EP2300.vector.GAPNode`. In `peersim.EP2300.message` three new messages were introduced: `ResponseTimeArriveMessage`, `UpdateVector`, `TimeOut`. The first two roughly equal `LOCALVAR` and `UPDATE & NEW` from the original technical report (see [1]).

Each node stores all its information on it's neighbors as `NodeStateVector` in a SortMap named `neighborList`. Details on the nodes response times are stored in the ArrayList `requestList`. To focus only on the response times in the current time window, every response time is assigned a time out.

For the implementation with rate control, every message send to the parent will trigger a decrease of the msg budget. As soon as the msg budget is lower or equal to 0, no more messages are sent out. The message budget is reset via a new control `peersim.EP2300.control.ResetMsgBudget`.

1.3. Results

(i) time series of f(t) and f(t) for r = {0.2,0.4,0.8} and {R1} from the first 5 minutes

(ii) time series of f(t) and f(t) for r = {0.2,0.4,0.8} and {R1} after the first 5 minutes

(iii) trade-off plots for both {R1,R2} (and all rate options?? 0.1,0.2,0.4,0.8,1.6)

(iv) density plots for r = {0.2, 0.4, 0.8} and {R1,R2}

2. Task

2.1. Pseudocode

In the following the pseudo code for extension 1 is presented. Different to the first task we do not use a message budget (or message rate). Instead an error budget is used. This error budget is assigned to all nodes. Only when the difference of the aggregate of the subtree to the last communicated version of the same exceeds the error budget, a message is sent.

Listing 2: Pseudocode for $\widetilde{f}(t)$ with error budget in P_1

```
messages:
[see listing 1]

objects and help functions:
[see listing 1]
A.difference_of_subtree # difference of last communicated value versus current
                        #  communicated value

constants:
DELAY := # fixed time window
INF := # really big number
ERROR_BUDGET = # fix number

global variables:

```

```
procedure GAP( )
    Neighbors := empty;
    ResponseTimes := empty;

    if v = root then
        [see listing 1]
    end if

    A.initiate()
    while true do
        read message;
        switch (message)
            [see listing 1]
        end switch
        if (A.difference_of_subtree() > ERROR_BUDGET) then
            Comm.send_to_parent(Neighbors, A.value)
        end if
    end while
end procedure
```

2.2. Implementation Details

Similar to the difference between the pseudocode of this and task 1 the difference in the code is just the replacement of a depletable message budget with a constant error budget.

3. Task

3.1. Pseudocode

The pseudocode for providing $\widetilde{g}(t)$ under P_1 is equal to the one provided in section 2. The pseudocode for P_2 is attached below.

4. foo

- compare performance for R_1, R_2
  - time series of f (t) and f(t) for r = { 0.2, 0.4, 0.8 } amd $\{R_1,\ R_2\}$ from the first 5 min
  - trade off plot for R_1, R_2
  - density plot for r = { 0.2, 0.4, 0.8 } and $\{R_1,\ R_2\}$

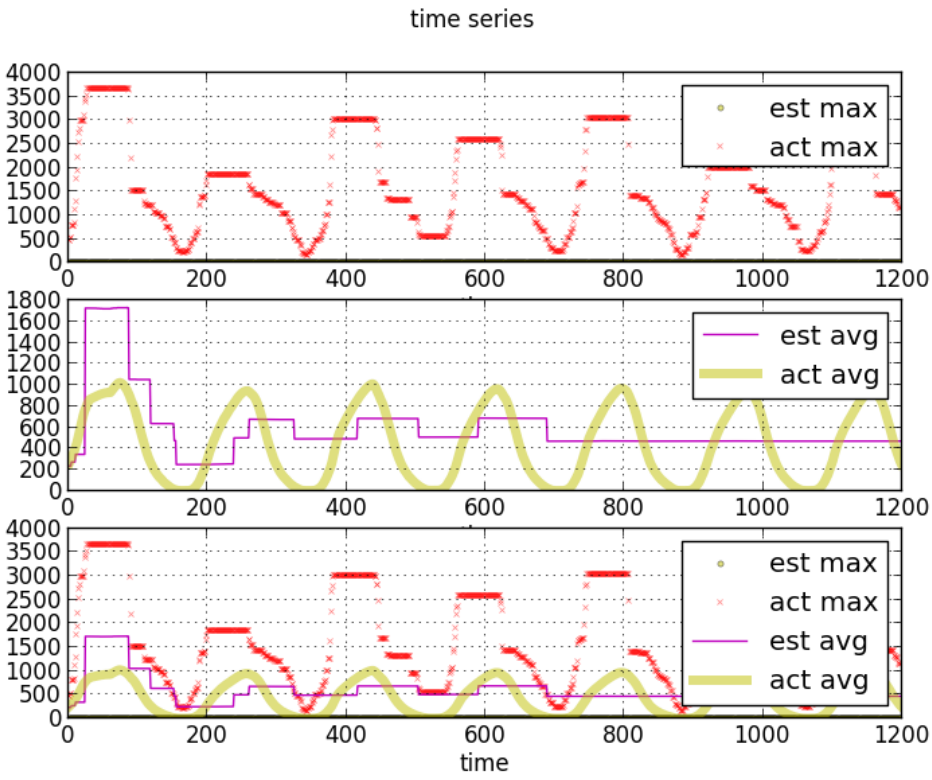

Abbildung 1: Time series plot

5. Task II

- pseudo code!
- implementation details
- compare performance for R_1, R_2
  - time series of f (t) and f(t) for r = { 0.2, 0.1, 0.05, 0.025, } amd $\{R_1, R_2\}$ from the first 5 min
  - trade off plot for R_1, R_2
  - density plot for r = { 0.1, 0.05, 0.025 } and $\{R_1, R_2\}$

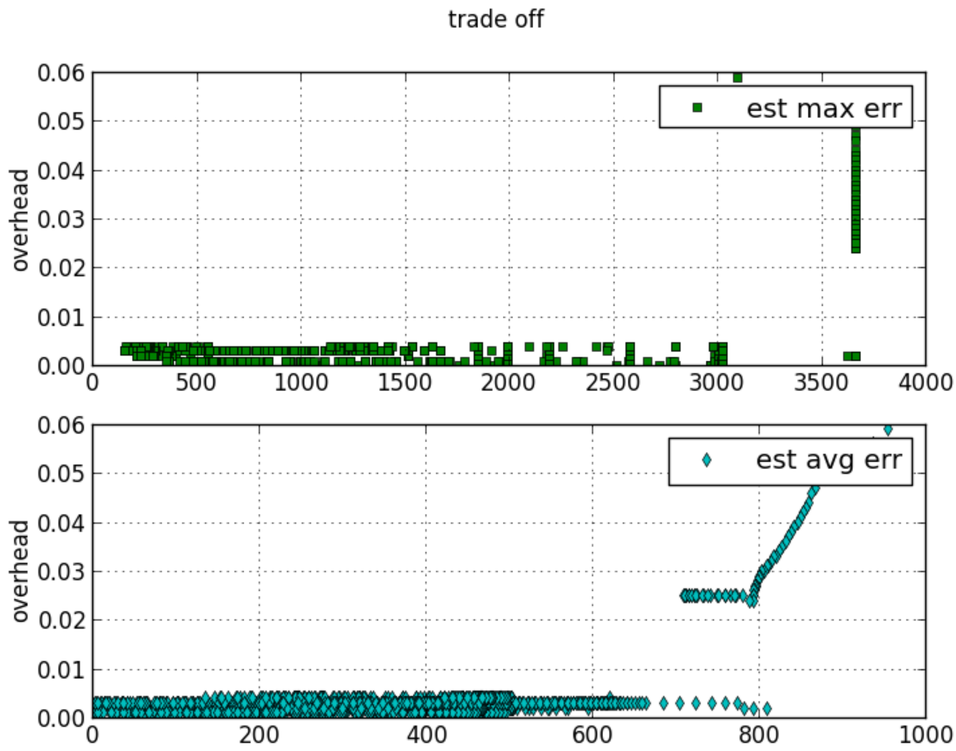

Abbildung 2: Trade off plot

6. Task III

- pseudo code!
- implementation details
- compare performance for R_1, R_2
  - time series of f (t) and f(t) for r = { 0.2, 0.1, 0.05, 0.025, } amd $\{R_1, R_2\}$ from the first 5 min
  - trade off plot for R_1, R_2
  - density plot for r = { 0.1, 0.05, 0.025 } and $\{R_1, R_2\}$

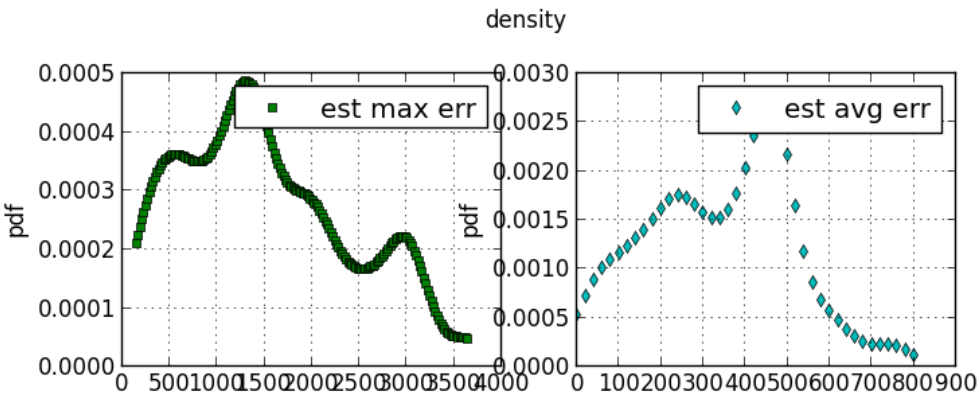

Abbildung 3: PDF plot

7. Summary

- Compare TaskI, II, III
- Compare R_1, R_2 globally

Literatur

[1] R. Stadler, "Protocols for distributed management," 2012.

A. Calculations

A.1. Distribution of error budget in aggregation via average

Definitions

ϵ	:= Local error budget of a node	≥ 0	
Φ	:= Overall error budget	≥ 0	
X	:= Previous sum of the response times in subtree	≥ 0	**Aim**
Y	:= Previous number of requests in subtree	≥ 0	
ΔY	:= Number of newly arrived reqeusts in subtree	≥ 0	

It is the aim of this proof to show how the error budget ϵ for every node i can be maximized (ϵ_{max}) while keeping the global error budget Φ constant. The aggregation operation in this case is averaging of the response times of all requests.

Proof

$$\Phi \geq \left| \frac{X + \Delta Y \cdot \epsilon}{Y + \Delta Y} - \frac{X}{Y} \right| \tag{1}$$

$$\epsilon \leq \begin{cases} \frac{(Y+\Delta Y)\Phi}{\Delta Y} + \frac{X}{Y} & \text{if } Y \neq 0 \\ \Phi & \text{if } Y = 0 \end{cases} \tag{2}$$

The task to find maximum value of ϵ can be transformed to finding minimum value of the right side of inequation 2.

When $Y \neq 0$ we have

$$\frac{Y + \Delta Y}{\Delta Y} > 1, \frac{X}{Y} \geq 0$$

thus

$$\frac{(Y + \Delta Y)\,\Phi}{\Delta Y} + \frac{X}{Y} \geq \Phi$$

When $Y = 0$, the minimum of right side is Φ, thus the minimum value of right side of inequeation is Φ.

It follows $\epsilon_{max} = \Phi$.

A.2. Distribution of error budget based on message rate

Definitions

Other than the definitions for problem 3a:

α	:= Factor of relation between error budget and request rate	≥ 0
δ	:= Number of newly arrived requests in a node per time unit	

Aim

The aim of this proof is to find the maximum factor α relating the request rate δ_i with the the error budget ϵ.

Proof

For every node i with the $S(i)$ denoting all nodes that are direct children of i in the network and i itself.

$$\left| \begin{array}{ll} \Phi \geq & \frac{X+\sum_{j \in S(i)} \epsilon_j \cdot \delta_j}{Y+\sum_{j \in S(i)} \delta_j} - \frac{X}{Y} \\ \epsilon_i = & \alpha \cdot \delta_i \end{array} \right| \tag{3}$$

Applying the same schema of problem 3a, we can have

$$\Phi \geq \frac{\sum_{j \in S(i)} \epsilon_j \cdot \delta_j}{\sum_{j \in S(i)} \delta_j} \tag{4}$$

substitute ϵ_i with $\alpha \cdot \delta_i$, we have:

$$\Phi \geq \frac{\sum_{j \in S(i)} \alpha \cdot \delta_j^2}{\sum_{j \in S(i)} \delta_j} \tag{5}$$

$$\alpha \leq \frac{\sum_{j \in S(i)} \delta_j}{\sum_{j \in S(i)} \delta_j^2} \cdot \Phi \tag{6}$$

Hence the max factor is:

$$\alpha_{max} = \frac{\sum_{j \in S(i)} \delta_j}{\sum_{j \in S(i)} \delta_j^2} \cdot \Phi \tag{7}$$

Thus error budget for node i is $\alpha_{max} \cdot \delta_i = \delta_i \frac{\sum_{j \in S(i)} \delta_j}{\sum_{j \in S(i)} \delta_j^2} \cdot \Phi$