Active Control with Backoff Algorithm for Reducing Broker Load in Smart Spaces

Andrey S. Vdovenko
Petrozavodsk State University (PetrSU)
Petrozavodsk, Russia
vdovenko@cs.karelia.ru

Abstract—Performance degradation on broker side and the subsequent possible failure is an important issue in smart spaces. One of the reasons for this problem is the increase in the number of requests to the broker beyond what it can handle. The broker can have own management tools to solve such problematic situations. More often, it is more difficult to implement relevant approaches to broker itself. Our suggestion is to implement additional mechanism on agents sides to reduce fault events. The mechanism is to add a timeout to the persistent request, the value of which is selected based on the current system state. For timeout determining we suggest active control with adaptive strategy and improve it by adding backoff algorithm.

I. INTRODUCTION

Smart spaces can be a perspective environments for deploying IoT applications due to the possible organization of dynamic detection and interaction between physical objects with use of the available context in their locations [1], [2]. We consider smart spaces deployed in localized resource-restricted IoT environments consumed by a group of mobile clients [3]. Such a smart space is typically associated with a physical spatial-restricted place equipped with a variety of devices, where everyday life objects, alongside traditional computers, become data processors and service constructors to their users, which use mobile devices for interactions.

A smart space requires a software platform, addressing interoperability across heterogeneous devices and providing multiple services. This paper considers the Smart-M3 platform, which provides open source middleware for implementation of the smart space concept in IoT environments [4]. The central component of the M3 architecture is the semantic information broker (SIB). Each SIB manages and shares a knowledge base with all the smart space participants. The knowledge base is semantic, in the form of the RDF triplestore, and operations with it through SSAP interaction protocol. In IoT, the broker can be hosted by heterogeneous devices as ordinary computers or embedded devices with low-capacity as single-board computers or wireless routers [5].

Large number of simultaneous requests can roughly reduce broker performance or even call its fault. The broker has own management tools to resolve such situation, but in some cases it is better to add additional mechanisms on the agent’s side. To resolve this issue from mobile client, they can perform a persistent query with timeouts. But there stands problem of what is an optimal way to calculate such timeouts. We suggest the use of active control with adaptive strategy [6], which could be improved by backoff algorithm. Using of active control allows to reduce broker load for different requests and backoff algorithm is intended to avoid simultaneous of the same requests from different clients.

The rest of the work is organized as follows. Section II describes Smart-M3 platform management tools for request pools. Section III shows the implementation of the suggested approach with use of active control for many clients with backoff algorithm. Section IV summarizes our current results.

II. REQUESTS POOL PROBLEM IN SMART SPACES

When working in smart spaces, there are problems associated with the overall performance of the broker and individual agents, most often mobile clients. Problems with the performance of the broker arise because of collisions of requests from many agents, that send them simultaneously or with small delays. To resolve such situations, the broker has operations to manage the processing of the incoming request flow [7]. Table I describes management tools that can be applied on the broker and client sides to solve overload situations. The broker can build a queue of requests in the order of their receiving [8]. As an optimization of this approach in the case of mobile clients, which rapidly send same requests, the broker could combine several queries into sets over the data area, and doesn’t perform unnecessary operations with the database and sends the result immediately [9].

The broker’s performance with high query intensity is strongly affected by the amount of information retrieved and the execution of additional operations (inserting and deleting triplets, SPARQL queries and subscription maintenance) because of which the queue begins to grow endlessly. In such cases, the broker can ignore the arrival of new requests, or do not perform queries that are more complicated in time and do not send out-of-date notifications [10]. But in this case, this strategy can adversely affect the work of the application system, so it is possible to set priorities performed requests for active agents, for example, when the highest priority is given to requests of service agents, and customer requests are performed in the best effort.

Problem of fault tolerance is vital issue of interests in related fields of distributed systems and client server communication networks [11], [12]. Increased loads leads to faults for them and a large bunch of problems in that cases cause by using Wireless LANs, because of higher losses percent due to collisions of radio signals and low-capacity routers. That is the same for the smart spaces issue. They use different approaches to achieve fault tolerance quality similar with the
Management on the side of the broker does not always solve the problems related to mobile clients, for example in the field of delivery of notifications. Therefore, in this case, the client can perform its own management, for example, by addition to the subscription operation of timeout checks, or completely replacing the subscription operation for independent active control of data updates. Fig. 1 demonstrates request pool to the broker from \( m \) mobile clients in case of active control. For simplicity, we assume that clients are the same and they perform same persistent requests. Each client sends \( n \) requests, denoted by \( R_i \). Each \( R_i \) request repeats after timeout value \( t_{ij} \), where \( j = 1, \ldots, m \) is the index of repeat, this relation can be denoted by \( t_{ij} = f(R_i, j) \). The broker receives requests and maintains queues of them and starts processing with intensity \( \lambda_{pr} \). Processing intensity mean that if the client request pool will grow with intensity greater than maximum queue length, broker would stop answering on requests even with successful subscription operation for independent active control on client side.

### III. Active control improved by backoff algorithm

The choice of the timeout value depends on the current situation in the system, i.e., the timeout is shorter when the data is updated more frequently and the load on the system small and bigger otherwise. To determine the value of the load on the broker, the query execution time can be used, since The average time for executing the query during a no load situation in the system, i.e., the timeout is shorter when the load is lower than maximum queue length.

![Fig. 1. Requests pool to the broker from \( m \) clients with same \( n \) requests of different types.](image)

**Algorithm 1** Active control with backoff algorithm for \( n \) persistent requests.

**Require:** initialization of active control on client side

1. for \( i = 0 \) to \( n \) do
2. \{ start thread for \( R_i \) performing \}
3. loop
4. doQuery(\( R_i \));
5. \( t_{i}^{act} = \text{activeControl}(R_i); \)
6. \( t_{i}^{bck} = \text{backoffTimeout}(R_i); \)
7. sleep(\( t_{i}^{act} + t_{i}^{bck} \));
8. end loop
9. end for

is known. Clients often fulfill their requests in parallel with similar parameters for determining the value of the timeout. In this case, it becomes necessary to determine the additional delay, aimed at reducing the number of simultaneously sent requests. The best mechanism for this purpose is the backoff algorithm. Its meaning is a successive increase in the timeout from which the size of the delay is randomly selected in the event of a collision of requests, for example, a significant increase in the execution time of the request. Due to that, total value of timeout for the request is equal to sum of active control timeout for \( i \) request and backoff algorithm timeout for client on current \( j \) repeat round, i.e., \( t_{ij} = t_{ij}^{act} + t_{ij}^{bck} \).

![Algorithm 1](image)

**TABLE I. MANAGEMENT TOOLS FOR OVERLOAD CONTROL**

<table>
<thead>
<tr>
<th>Host</th>
<th>Tool</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIB</td>
<td>FIFO processing</td>
<td>Requests pool processed in order of receiving</td>
</tr>
<tr>
<td></td>
<td>Queries caching</td>
<td>Caching same queries in memory without interaction with database</td>
</tr>
<tr>
<td></td>
<td>Request reject</td>
<td>Requests queue have length based on broker host capacity. Algorithm rejects requests after they number are bigger than maximum queue length</td>
</tr>
<tr>
<td>Clients</td>
<td>Active control strategy</td>
<td>Long-term requests performed with timeout. Timeout value calculation is based on current system situation</td>
</tr>
</tbody>
</table>
Algorithm 2 Active control algorithm for n persistent requests.

function activeControlTimeout(Ri):
1: \( k_j = \text{getQueryLosses}(R_i) \); \{receive losses number\}
2: calculateEstimates(); \{recalculate estimates\}
3: if \( k_j = 0 \) then
4: \( \tau_{i,j}^\text{act} = \frac{n}{(j-1)} + N \times \delta \);
5: if \( T - \tau_{i,j}^\text{act} \leq 0 \) then
6: \{use previous timeout value to avoid losses\}
7: \( \tau_{i,j}^\text{act} = \tau_{i,j-1}^\text{act} \);
8: end if
9: else
10: \( \tau_{i,j}^\text{act} = \tau_{i,j-1}^\text{act} / \alpha \);
11: end if
12: return \( \tau_{i,j}^\text{act} \);

Algorithm 3 Backoff algorithm for clients.

function backoffTimeout(R_i):
1: \( j++ \);
2: \{if backoff was reset on previous round\}
3: if \( t_{bck}^j = 0 \) then
4: \( t_{bck}^j = \text{minTimeout} \);
5: end if
6: \{determine performance level\}
7: duration = getLastErrorDuration(R_i);
8: if duration \leq \text{getAverageQueryDuration}(R_i) then
9: \( t_{bck}^j = 0 \);
10: return \( t_{bck}^j \);
11: end if
12: \{calculate backoff timeout\}
13: \( t_{bck}^j = \text{min}(t_{bck}^{j-1}, \text{factor}, \text{maxTimeout}) \);
14: \( t_{bck}^j = t_{bck}^j + \text{variation}(t_{bck}^j \times \text{seed}) \);
15: return \( t_{bck}^j \);

own calculations of timeout values denoted before by \( \tau_{i,j}^\text{act} \),
where \( i \) is the index of request in the range of one client.
Algorithm 2 show implementation of adaptive strategy correspondingly with (1) for \( n \) persistent requests with use of analytical estimates. For each request starts a separate thread to parallel execution of queries. After each query processing, we calculate estimates to reflect on the current system situation. The number of consecutive grows \( N \) is used for a fast increase of \( \tau_{i,j} \) to its high bound. Check current \( \tau_{i,j}^\text{act} \) values and if they are bigger than the expected length \( T \), we use previous timeout value.

b) Backoff timeout: Adaptive strategy in active control has limitations in the case of many parallel clients. That is because of the same parameters for requests across all clients, which mean that requests will be performed in the same time and this will lead to performance degradation. To spread requests on timeline suggested use backoff algorithm for each request on detecting performance degradation.

The main idea in the implementation of the backoff algorithm for smart spaces is event detection when we need to use it. The most simplest way is a measure of average round trip time (RTT), i.e., we know the duration of the operation in most of cases and if the operation takes more time we need to use the backoff algorithm to improve this by reducing simultaneous request number. If first use doesn’t give success, then we need to repeat backoff algorithm until RTT gets close to average.

Standard backoff algorithm can be described by following equation:

\[
t_{bck}^j = \min(t_{bck}^{j-1} \times \text{factor}, \text{maxTimeout}),
\]

where \( \text{factor} \geq 2 \) and \( \text{maxTimeout} \leq 5 \) \times average query duration, that stands for stopping increase backoff value after 5 rounds. This algorithm is currently in use in all 802.11 standards.

To add additional randomize for backoff value selection can be used variation described with following:

\[
t_{bck}^j = t_{bck}^j + \text{variation}(t_{bck}^j \times \text{seed}),
\]

where variation is random function that returning value, which follows normal distribution and seed is for random initialization.

Algorithm 3 shows how the backoff algorithm can be implemented for active control to decrease simultaneous requests with use of equations (2) and (3). Function store value of previously calculated timeout and reset it to zero on performance stabilization. The constant \( \text{minTimeout} \) is the start value (e.g., 100 ms). The constant \( \text{maxTimeout} \) is upper bound to timeout (e.g., 1 minute). The constant factor is coefficient of timeout grow (e.g., 2). The constant seed is used for random fluctuations and can be equal to system time. Duration of last query is less or equal to average value we stop backoff algorithm by setting its timeout to zero.

IV. CONCLUSION

We describe our approach to resolve problem of high request pool of mobile clients to broker. Our suggestion is to use active control for persistent queries with improving it by backoff algorithm to avoid simultaneous requests. Improvements are achieved by reducing the number of requests to the broker, as well as in the event of a decrease in the intensity of broker processing, clients begin to be distributed on a timeline to allow the broker to stabilize its work. The direction of our future work is evaluation of the proposed solution with the use of an experimental system consisted of services and clients.

ACKNOWLEDGMENT

This research is financially supported by the Ministry of Education and Science of Russia within project # 2.5124.2017/8.9 of the basic part of state research assignment for 2017–2019. The work is implemented within the Government Program of Flagship University Development for Petrozavodsk State University in 2017–2021.

REFERENCES


