Branch: **master** Find file Copy path

paxosstore / paxoskv / core / **pins_wrapper.h**

dengoswei init add;

aca2d18 on Aug 25, 2017

1 contributor

Raw Blame History

237 lines (170 sloc) 5.32 KB

```
1
2   /*
3   * Tencent is pleased to support the open source community by making PaxosStore available.
4   * Copyright (C) 2017 THL A29 Limited, a Tencent company. All rights reserved.
5   * Licensed under the BSD 3-Clause License (the "License"); you may not use this file except in compliance with the Licens
6   * https://opensource.org/licenses/BSD-3-Clause
7   * Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an
8   */
9
10
11
12  #pragma once
13
14  #include <unistd.h>
15  #include <stdint.h>
16  #include <tuple>
17  #include <map>
18  #include <string>
19  #include <memory>
20  #include <chrono>
21  #include "cutils/id_utils.h"
22  #include "paxos.pb.h"
23
24
25  namespace paxos {
26
27  // private
28  enum class PropState : uint8_t {
29      NIL = 0,
30      PREPARE = 1,
31      WAIT_PREPARE = 2,
32      ACCEPT = 3,
33      WAIT_ACCEPT = 4,
34      CHOSEN = 5,
35
36          PROP_FROZEN = 6,
37
38          // RELOAD_
39  };
40
41  enum {
42          MAX_PROP_CNT = 3,
43  };
44
45  class PInsAliveState {
46
47      // test interface
48  public:
49      const Entry* TestProposingValue() const {
50          return proposing_value_.get();
51      }
52
```

```
    uint64_t TestProposedNum() const {
        return prop_num_gen_.Get();
    }

    const std::map<uint8_t, bool> TestRspVotes() const {
        return rsp_votes_;
    }

    PropState TestPropState() const {
        return prop_state_;
    }

    uint64_t TestMaxAcceptedHintNum() const {
        return max_accepted_hint_num_;
    }

    uint64_t TestMaxHintNum() const {
        return max_hint_num_;
    }

        std::unique_ptr<PInsAliveState> TestClone();

    // copy construct for test:
    PInsAliveState(const PInsAliveState& other) = delete;

    PInsAliveState& operator=(const PInsAliveState& other) = delete;

    // end of test interface
public:

    PInsAliveState(
            const std::string& key,
                        uint64_t index,
                        uint64_t proposed_num);

        ~PInsAliveState();

    std::tuple<bool, MessageType>
        Step(const Message& msg, PaxosInstance& pins_impl);

    bool HasProposingValue() const {
        return nullptr != proposing_value_;
    }

        std::unique_ptr<paxos::Entry> ClearProposingValue() {
                auto proposing_value = std::move(proposing_value_);
                assert(nullptr == proposing_value_);
                return proposing_value;
        }

    bool IsChosen() const {
        return PropState::CHOSEN == prop_state_;
    }

    void MarkChosen();

    int GetNotifyFD() const {
        assert(0 <= pipes_[0]);
        assert(0 <= pipes_[1]);
        return pipes_[0];
    }

        void SendNotify() const;

        PropState GetPropState() const {
                return prop_state_;
```

```
        }

        uint64_t GetIndex() const {
                return index_;
        }

    const std::string& GetKey() const {
        return key_;
    }
        bool IsLocalProposeNum(uint64_t prop_num) const {
                return prop_num_gen_.IsLocalNum(prop_num);
        }

        uint64_t GetProposedNum() const {
                return prop_num_gen_.Get();
        }

        uint64_t GetActiveBeginProposedNum() const {
                return active_begin_prop_num_;
        }

        const paxos::Entry& GetProposingValue() const {
                assert(nullptr != proposing_value_);
                return *proposing_value_;
        }

private:
    PropState stepPrepareRsp(
            uint8_t peer_id,
            uint64_t peer_promised_num,
            uint64_t peer_accepted_num,
            const Entry* peer_accepted_value);

    PropState stepAcceptRsp(
            uint8_t peer_id,
            uint64_t peer_accepted_num,
            bool is_fast_accept_rsp);

    PropState stepTryPropose(
            uint64_t hint_proposed_num,
            const paxos::Entry& try_proposing_value);

    PropState stepBeginPropose(
            uint64_t hint_proposed_num,
            const paxos::Entry& proposing_value);

    PropState beginPreparePhase(PaxosInstance& pins_impl);

    PropState beginAcceptPhase(PaxosInstance& pins_impl);

    std::tuple<bool, MessageType>
        updatePropState(PropState next_prop_state, PaxosInstance& pins_impl);

private:
    static const int major_cnt_ = 2; // 2 is major in 3-size group
    std::string key_;
        uint64_t index_ = 0;

        uint16_t active_prop_cnt_ = 0;
        uint64_t active_begin_prop_num_ = 0;

    cutils::PropNumGen prop_num_gen_;
    PropState prop_state_ = PropState::NIL;

    uint64_t max_accepted_hint_num_ = 0ull;
    uint64_t max_hint_num_ = 0ull;
```

```
185
186     std::map<uint8_t, bool> rsp_votes_;
187     std::unique_ptr<Entry> proposing_value_;
188
189     // TODO
190     // pipes_[0]: poll
191     // pipes_[1]: notify: write or close
192     int pipes_[2] = {-1, -1};
193 };
194
195
196 class PInsWrapper {
197
198 public:
199     // test function
200     const PaxosInstance* TestGetPaxosInstance() const {
201         return &pins_impl_;
202     }
203     // end of test function
204
205 public:
206     PInsWrapper(
207             PInsAliveState* pins_state, PaxosInstance& pins_impl);
208
209     std::tuple<int, bool, std::unique_ptr<Message>> Step(const Message& msg);
210
211     bool IsChosen() const {
212         return pins_impl_.chosen();
213     }
214
215 private:
216     void markChosen();
217
218     std::tuple<int, bool, std::unique_ptr<Message>>
219         stepChosen(const Message& msg);
220
221     std::tuple<int, bool, std::unique_ptr<Message>>
222         stepNotChosen(const Message& msg);
223
224     std::unique_ptr<Message>
225         produceRsp(const Message& msg, MessageType rsp_msg_type);
226
227
228 private:
229     PInsAliveState* pins_state_;
230     PaxosInstance& pins_impl_;
231 };
232
233
234 } // namespace paxos
235
236
```