Connection Management

2024 ATS Spring Summit

Masaori Koshiba (masaori@apache.org)

1. Connection Throttling

2. Better Connection Control via TS API

1. Connection Throttling (Issue #9681)

> Under a very large amount of connects, this could cause a long event handler time and create a large backlog of events, causing timeouts and other issues.

Recent Changes (Milestone: 10.0.0)

- Allow configurable number of accepts per event loop #10098
- Update ACCEPT thread configurations #10687
- proxy.config.net.per_client.max_connections_in #10688

Related Configs

```
net:
  # Server::listen()
  listen_backlog

  # NetAccept::do_blocking_accept()
  additional_accepts
  throttle_delay
  connections_throttle
  per_client.max_connections_in

  # NetHandler::manage_keep_alive_queue()
  max_connections_in
```

```
// src/iocore/net/Connection.cc
int
get_listen_backlog()
{
  int listen_backlog;

  REC_ReadConfigInteger(listen_backlog, "proxy.config.net.listen_backlog");
  return (0 < listen_backlog && listen_backlog <= 65535) ? listen_backlog : ats_tcp_somaxconn();
}

int
Server::listen(bool non_blocking, const NetProcessor::AcceptOptions &opt)
{
  ...
  if ((res = safe_listen(fd, get_listen_backlog())) < 0) {
    goto Lerror;
  }
```

```
// src/iocore/net/UnixNetAccept.cc
int
NetAccept::do_blocking_accept(EThread *t)
{
  ...
  int additional_accepts = NetHandler::get_additional_accepts();      ///< proxy.config.net.additional_accepts

  do {
    if ((res = server.accept(&con)) < 0) {
      int seriousness = accept_error_seriousness(res);
      switch (seriousness) {
      case 0:
        // bad enough to warn about
        check_transient_accept_error(res);
        safe_delay(net_throttle_delay);                               ///< proxy.config.net.throttle_delay
        return 0;
      case 1:
        // not so bad but needs delay
        safe_delay(net_throttle_delay);                               ///< proxy.config.net.throttle_delay
        return 0;
      ...
    }
    // check for throttle
    if (check_net_throttle(ACCEPT)) {                                 ///< proxy.config.net.connections_throttle
      check_throttle_warning(ACCEPT);
      // close the connection as we are in throttle state
      con.close();
      Metrics::Counter::increment(net_rsb.connections_throttled_in);
      continue;
    }
    std::shared_ptr<ConnectionTracker::Group> conn_track_group;
    if (!handle_max_client_connections(con.addr, conn_track_group)) { ///< proxy.config.net.per_client.max_connections_in
      con.close();
      continue;
    }
    ...
  } while (count < additional_accepts);
```

```
// src/iocore/net/NetHandler.cc
void
NetHandler::manage_keep_alive_queue()
{
  ...
  for (NetEvent *ne = keep_alive_queue.head; ne != nullptr; ne = ne_next) {
    ne_next = ne->keep_alive_queue_link.next;
    _close_ne(ne, now, handle_event, closed, total_idle_time, total_idle_count);

    total_connections_in = active_queue_size + keep_alive_queue_size;
    if (total_connections_in <= max_connections_per_thread_in) {
      break;
    }
  }
```

Origin Server Side Connection Throttling

```
http:
  server_max_connections
  per_server.connection.max
```

- per_server.connection.max: CONNECT requests #9182
- Reply with a 503 response when per_server.connection.max is exceeded #9121

```
// src/proxy/http/HttpSM.cc
void
HttpSM::do_http_server_open(bool raw, bool only_direct)
{
  ...
  // Check to see if we have reached the max number of connections.
  // Atomically read the current number of connections and check to see
  // if we have gone above the max allowed.
  if (t_state.http_config_param->server_max_connections > 0) {
    if (Metrics::Gauge::load(http_rsb.current_server_connections) >= t_state.http_config_param->server_max_connections) {
      httpSessionManager.purge_keepalives();
      // Eventually may want to have a queue as the origin_max_connection does to allow for a combination
      // of retries and errors.  But at this point, we are just going to allow the error case.
      t_state.current.state = HttpTransact::CONNECTION_ERROR;
      call_transact_and_set_next_state(HttpTransact::HandleResponse);
      return;
    }
  }
```

```
// src/proxy/http/HttpSM.cc
void
HttpSM::do_http_server_open(bool raw, bool only_direct)
{
  ...
  // Check to see if we have reached the max number of connections on this upstream host.
  if (t_state.txn_conf->connection_tracker_config.server_max > 0) {
    auto &ct_state       = t_state.outbound_conn_track_state;
    auto ccount          = ct_state.reserve();
    int const server_max = t_state.txn_conf->connection_tracker_config.server_max;
    if (ccount > server_max) {
      ct_state.release();

      ink_assert(pending_action.empty()); // in case of reschedule must not have already pending.

      ct_state.blocked();
      Metrics::Counter::increment(http_rsb.origin_connections_throttled);
      ct_state.Warn_Blocked(server_max, sm_id, ccount - 1, &t_state.current.server->dst_addr.sa,
                            debug_on && dbg_ctl_http.on() ? "http" : nullptr);
      send_origin_throttled_response();
      return;
    } else {
      ct_state.Note_Unblocked(&t_state.txn_conf->connection_tracker_config, ccount, &t_state.current.server->dst_addr.sa);
    }

    ct_state.update_max_count(ccount);
  }
```

2. Better Connection Control via TS API

The rate_limit plugin (as a global plugin) counts current connections and apply rate limiting. However, it doesn't know state of connections. Because no such TS APIs.

- e.g.

Apply rate limiting on "active" connections. If there're too many active connections, park some of them into the keep-alive queue (or some pool) for a while.