Aplicação Cliente/Servidor usando KAFKA em C

Server

```
1  void run_server(const struct context *const ctx __UNUSED) {
2      const char *broker_id = "127.0.0.1:9092", *group_id = "server", *consumer_topic = "request", *producer_topic;
3      /* ... */
4
5      rd_kafka_conf_set_rebalance_cb(rkc, rebalance_callback);
6      rd_kafka_conf_set_opaque(rkc, &rebalanced);
7
8      if (RD_KAFKA_CONF_OK != rd_kafka_conf_set(rkc, "bootstrap.servers", broker_id, err_str, sizeof(err_str)))
9          die(EXIT_FAILURE, NOERR, "Failed to set bootstrap.servers: %s", err_str);
10
11     if (RD_KAFKA_CONF_OK != rd_kafka_conf_set(rkc, "group.id", group_id, err_str, sizeof(err_str)))
12         die(EXIT_FAILURE, NOERR, "Failed to set group.id: %s", err_str);
13
14     if (NULL == (rk_p = rd_kafka_new(RD_KAFKA_PRODUCER, rd_kafka_conf_dup(rkc), err_str, sizeof(err_str))))
15         die(EXIT_FAILURE, NOERR, "Failed to create new producer: %s", err_str);
16     else
17         kafka_start_producer_polling(rk_p);
18
19     if (NULL == (rk_c = rd_kafka_new(RD_KAFKA_CONSUMER, rkc, err_str, sizeof(err_str))))
20         die(EXIT_FAILURE, NOERR, "Failed to create new consumer: %s", err_str);
21     else {
22         rkc = NULL;
23         rd_kafka_poll_set_consumer(rk_c);
24     }
25
26     if (RD_KAFKA_RESP_ERR_NO_ERROR != (r_err = kafka_subscribe(rk_c, consumer_topic, RD_KAFKA_PARTITION_UA)))
27         die(EXIT_FAILURE, NOERR, "Failed to subscribe to %s: %s", consumer_topic, rd_kafka_err2str(r_err));
28
29     while (!rebalanced)
30         kafka_consumer_poll(rk_c, 200);
31     /* ... */
```

Server - Loop

```
    for (;;) {
        if (NULL == (rkm = kafka_consumer_poll(rk_c, 200)))
            continue;
        else if (RD_KAFKA_RESP_ERR_NO_ERROR != rkm->err) {
            log_print(ERROR, "Failed to consume from \"%s\": %s", consumer_topic, rd_kafka_message_errstr(rkm));
        } else {
            log_print(NOISY, "Message %"PRId64" received from topic \"%s\"", rkm->offset, rd_kafka_topic_name(rkm->rkt));

            if (RD_KAFKA_RESP_ERR_NO_ERROR != (r_err = rd_kafka_message_headers(rkm, &rkm_headers))) {
                log_print(ERROR, "Failed to get message header: %s", rd_kafka_err2str(r_err));
            } else if (RD_KAFKA_RESP_ERR_NO_ERROR != (r_err = rd_kafka_header_get_last(rkm_headers, "REPLY_TOPIC", (const void **)
&producer_topic, &header_value_size))) {
                log_print(ERROR, "Failed to read message header: %s", rd_kafka_err2str(r_err));
            } else if (rkm->payload && rkm->len == sizeof(struct request)) {
                reply = calc(*((struct request *) rkm->payload));

                r_err = rd_kafka_producev(rk_p, RD_KAFKA_VTYPE_TOPIC, producer_topic, RD_KAFKA_VTYPE_MSGFLAGS, RD_KAFKA_MSG_F_COPY,
                                          RD_KAFKA_VTYPE_VALUE, &reply, sizeof(reply), RD_KAFKA_VTYPE_END);

                if (RD_KAFKA_RESP_ERR_NO_ERROR != r_err) {
                    log_print(ERROR, "Failed to delivery message to topic \"%s\": %s", producer_topic, rd_kafka_err2str(r_err));
                } else {
                    log_print(NOISY, "Enqueued message for topic \"%s\"", producer_topic);
                }
            }
        }

        rd_kafka_message_destroy(rkm);
    }
}
```

Client

```
static __always_inline const struct kafka_handle *kafka_start() { /* ... */
   kh->client_id = uuid_gen();  kh->client_id_len = strlen(kh->client_id);

   rd_kafka_conf_set_rebalance_cb(rkc, rebalance_callback);   rd_kafka_conf_set_opaque(rkc, &rebalanced);

   if (RD_KAFKA_CONF_OK != rd_kafka_conf_set(rkc, "bootstrap.servers", broker_id, err_str, sizeof(err_str)))
        die(EXIT_FAILURE, NOERR, "Failed to set bootstrap.servers: %s", err_str);

   if (RD_KAFKA_CONF_OK != rd_kafka_conf_set(rkc, "group.id", group_id, err_str, sizeof(err_str)))
        die(EXIT_FAILURE, NOERR, "Failed to set group.id: %s", err_str);

   if (NULL == (kh->rk_p = rd_kafka_new(RD_KAFKA_PRODUCER, rd_kafka_conf_dup(rkc), err_str, sizeof(err_str))))
        die(EXIT_FAILURE, NOERR, "Failed to create new producer: %s", err_str);
   else kafka_start_producer_polling(kh->rk_p);

   if (NULL == (kh->rk_c = rd_kafka_new(RD_KAFKA_CONSUMER, rkc, err_str, sizeof(err_str))))
        die(EXIT_FAILURE, NOERR, "Failed to create new consumer: %s", err_str);
   else {
        rkc = NULL;  rd_kafka_poll_set_consumer(kh->rk_c);
   }

   if (NULL == (rkt_c = rd_kafka_topic_new(kh->rk_p, kh->client_id, NULL)))
        die(EXIT_FAILURE, NOERR, "Failed to create new topic: %s", rd_kafka_err2str(rd_kafka_last_error()));
   else {
        if (RD_KAFKA_RESP_ERR_NO_ERROR != (r_err = rd_kafka_produce(rkt_c, RD_KAFKA_PARTITION_UA, RD_KAFKA_MSG_F_COPY, "0123456789", 10, NULL, 0, NULL)))
             die(EXIT_FAILURE, NOERR, "Failed to delivery message to topic \"%s\": %s", kh->client_id, rd_kafka_err2str(r_err));

        rd_kafka_topic_destroy(rkt_c); sleep(1);
   }

   if (RD_KAFKA_RESP_ERR_NO_ERROR != (r_err = kafka_subscribe(kh->rk_c, kh->client_id, RD_KAFKA_PARTITION_UA)))
        die(EXIT_FAILURE, NOERR, "Failed to subscribe to \"%s\": %s", kh->client_id, rd_kafka_err2str(r_err));

   while (!rebalanced) kafka_consumer_poll(kh->rk_c, 200);

   return kh;
}
```

Client - Request

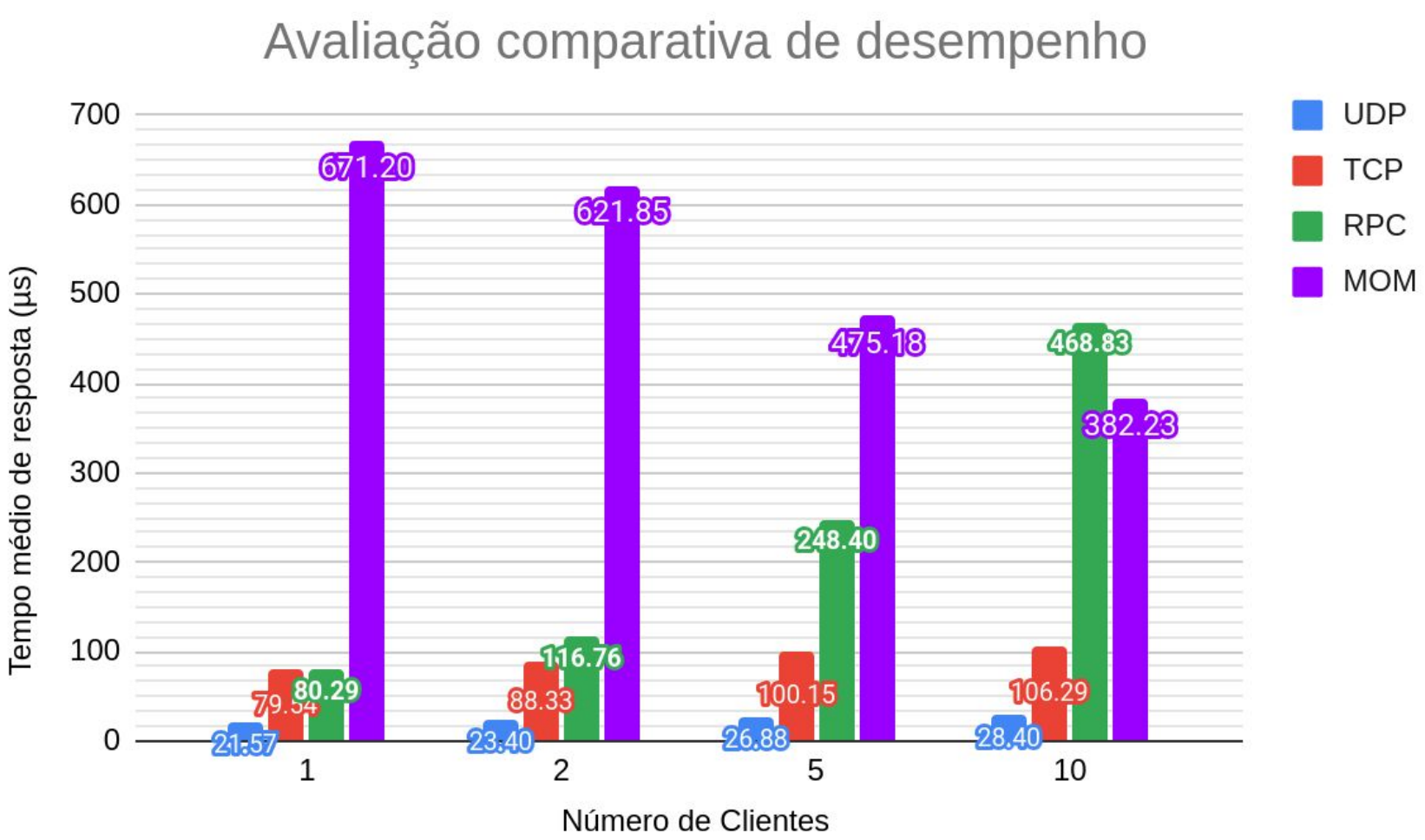
```c
void run_client(const struct context *const ctx __UNUSED) {
    struct request request;
    const struct kafka_handle *kh = kafka_start();

    log_print(INFO, "Started");

    for (;;) {
        request.a = htons((rand() % UINT16_MAX) + 1);
        request.b = htons((rand() % UINT16_MAX) + 1);
        request.op = (rand() % 2) == 0 ? ADD : SUB;

        send_request(request, kh);
    }
}
```

Client - Send Request

```c
static __always_inline bool send_request(const struct request request, const struct kafka_handle *kh) { /* ... */
   r_err = rd_kafka_producev(kh->rk_p, RD_KAFKA_VTYPE_TOPIC, producer_topic, RD_KAFKA_VTYPE_PARTITION, RD_KAFKA_PARTITION_UA,
                             RD_KAFKA_VTYPE_MSGFLAGS, RD_KAFKA_MSG_F_COPY, RD_KAFKA_VTYPE_VALUE, &request, sizeof(request),
                             RD_KAFKA_VTYPE_HEADER, "REPLY_TOPIC", kh->client_id, kh->client_id_len, RD_KAFKA_VTYPE_END);

   if (RD_KAFKA_RESP_ERR_NO_ERROR != r_err)
       log_print(WARN, "Failed to delivery message to topic \"%s\": %s", producer_topic, rd_kafka_err2str(r_err));
   else
       log_print(NOISY, "Enqueued message for topic \"%s\"", producer_topic);

   if (NULL == (rkm = kafka_consumer_poll(kh->rk_c, 5000))) {
       log_print(WARN, "No reply"); return false;
   } else if (RD_KAFKA_RESP_ERR_NO_ERROR != rkm->err)
       log_print(WARN, "Failed to consume from \"%s\": %s", kh->client_id, rd_kafka_message_errstr(rkm));
   else {
       log_print(NOISY, "Message %"PRId64" received from topic \"%s\"", rkm->offset, rd_kafka_topic_name(rkm->rkt));

       if (NULL != rkm->payload && sizeof(int_32) == rkm->len) {
           reply = *((int_32 *) rkm->payload);

           if (reply != calc(request))
               die(EXIT_FAILURE, NOERR, "Result: %.5d %c %.5d != %.5d", ntohs(request.a), request.op, ntohs(request.b), (int_32) ntohl((uint_32) reply));
           else
               log_print(NOISY, "Result: %.5d %c %.5d = %.5d", ntohs(request.a), request.op, ntohs(request.b), (int_32) ntohl((uint_32) reply));

           rd_kafka_message_destroy(rkm); return true;
       }
   }

   rd_kafka_message_destroy(rkm); return false;
}
```

Client - Benchmark

```
1   uint_8 run_client_benchmark(const struct context *const ctx) { /* ... */
2      struct request request = { .a = 10000, .b = 15000, .op = ADD };
3
4      for (uint_16 i = 0; i < 20; ++i)
5          send_request(request, kh);
6
7      for (uint_16 i = 0; i < ctx->benchmark_num; ++i) {
8          request.a = htons((rand() % UINT16_MAX) + 1); request.b = htons((rand() % UINT16_MAX) + 1);
9
10         begin = clock();
11         send_request(request, kh);
12         total_time += (times[i] = (double) (clock() - begin) / (CLOCKS_PER_SEC / 1000000.0));
13     }
14
15     avg = total_time / ctx->benchmark_num;  min = times[0];
16
17     for (uint_16 i = 0; i < ctx->benchmark_num; ++i) {
18         max = max > times[i] ? max : times[i];  min = min < times[i] ? min : times[i];
19         mdev += pow(times[i] - avg, 2);
20     }
21
22     mdev = sqrt(mdev / ctx->benchmark_num);
23
24     for (uint_16 i = 0; i < ctx->benchmark_num; ++i)
25         printf("%.0f\n", times[i]);
26
27     printf("min/avg/max/mdev = %.3f/%.3f/%.3f/%.3f µs\n", min, avg, max, mdev);
28
29     return EXIT_SUCCESS;
30 }
```


Avaliação comparativa de desempenho
Tempo médio de resposta (µs)
700
600
500
400
300
200
100
0
671.20
621.85
475.18
468.83
382.23
248.40
116.76
80.29
79.54
88.33
100.15
106.29
21.57
23.40
26.88
28.40
1
2
5
10
Número de Clientes
UDP
TCP
RPC
MOM