International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Command Transfer Protocol (CTP) For Distributed or Parallel Computation

John[1], Rajiv Sharma[2]

[1]M. Tech Scholar, Baba Mastnath University, Asthal Bohar, Rohtak, Haryana-124001, India

[2]Assistant Professor, CSE Deptt , Baba Mastnath University, Asthal Bohar, Rohtak, Haryana-124001, India

Abstract: ***Distribution of user-defined toolboxes and rapid prototyping of many coarse-grained parallel applications can now be done with a single easy-to-use command. The implementation is made available as a suite of three toolboxes, collectively described as mGrid, with client, master and worker implementations. Commands Transfer Protocol (CTP) is a new protocol and API for computational clusters. It meant as a replacement for both TCP and for MPI, PVM, and other high performance computing (HPC) APIs. CTP is a transport level API and thus replaces TCP. We are using CTP for distributed or parallel computation for file transfer***

Keywords: CPT (Command Transfer Protocol), Distributed Computation, Parallel computation

1. Introduction

Cluster is a union of workstations, which is formed for some definite purposes. Computational cluster is a cluster, which is built for heavy computations. It is a specific system that asserts special requirements for network functionality. Main properties of the networking mechanism for a good quality cluster are:

- Fast data interchange.
- Reliable data transfer.
- Broadcasting support.

As usual, all workstations inside some net take part in the computational experiment, so broadcasting makes controlling much easier.

- Huge data blocks interchange support. Sometimes, for example, initial conditions of experiment can be represented by such a block.
- Peer-to-peer networking. Any workstation can be the data source and the data destination, so they all are clients and servers simultaneously.

Majority of parallel computing using standard networking protocol TCP/IP [1], there are a lot of disadvantages of using this protocol:

- Low speed of data interchange. The "reliability" and "universality" of TCP has a lot of overhead charges. This protocol is a general-purpose one, so it is suitable for working in such unstable matter as Internet, but in a constant (or quite constant) system, which was developed for computations, it is possible to get more benefits.
- TCP does not support broadcasting. UDP does, but it is not reliable and the size of UDP datagram is limited by 65467 bytes [1].
- Ideology of logical channel creation before data interchange is redundant for cluster computations. Firstly because cluster, as usual, is a well tuned, good working net. Secondly because, some strategies of cluster computing lead to disordered interchange between workstations.
- TCP is a stream-based protocol, but, for determined tasks, bounded blocks interchange is preferable, because it allows to say definitely, when all data, necessary for further operations, have arrived.

CTP is a protocol that is to satisfy needs of arbitrary tasks, which, need support of rapid messages interchange and which can start heavy computations as a reaction for message receipt. Despite the fact that the letter "P" from its name means "protocol", it is not just a specification.

2. Command Transfer Protocol (CTP)

CTP is a transport level API and runs on top of UDP/IP. Transfer is reliable and supports broadcasting. CTP is twice faster than TCP while working with normal commands and not very large commands that can be brought by several packets. Packets larger than UDP's limit of 65400 bytes will be segmented by the sender with each segment sent one at a time. The receiver will reassemble the segments and only notify the application when the entire "large command" has been received. Each node is both a client and server and can send and receive commands. The basic abstraction used in CTP is "command". Command is an order from somebody to someone to do something (in most cases, workstations in clusters are communicating exactly in this way) or the response for such an order. From the last sentence, it is possible to conclude that a command is characterized by the following parameters:

- "Somebody" – sender
- "Someone" – recipient
- "Something" - command's description.

So, first of all, it is needed to define the sender and the recipient somehow. For this purpose, IP addresses will be used. The reason is that IP is used extremely widely and it fully satisfies the requirements (gives unique identifiers to all workstations). Commands will be identified by integer numbers.

In terms of the discussed protocol words, "command" and "message" are, actually, synonyms. "Command" is

"message", but not always vice versa.

CTP needs to satisfy the cluster networking requirements, listed above. The way in which this will be achieved follows (in the same order as in the introduction):

1) For incrementing the speed of interchange, UDP will be used as the basis of the protocol. Moreover, the usage of UDP, without going down to raw networking, will save the user in future from additional problems with the protocol support toolkit's installation.
2) Reliability of data interchange is to be implemented. Each sent packet will be stored until the recipient has not confirmed the receipt of the data. To maintain this mechanism, packets are to be provided with identifiers. Identification will be performed by assigning integer numbers on the sender-side. These IDs cannot be unique in general, but are to be unique for each sender.
3) Broadcasting support is one more argument to use UDP as the basic protocol.
4) Huge data interchange support is to be implemented. If a message that is greater than some limit (65400 bytes, by default) is going to be sent, then it is to be divided into smaller parts. These parts will be enumerated and sent to the recipient separately, one by one. On the recipient's side, they will be united to arrange the initial command. An important note is that the recipient application will get information about the command's arrival only after all its parts have been received. Such commands will be named as "large commands", but on practice, the majority of commands are "normal" (need a single packet for its transfer).
5) For peer-to-peer interchange, CTP's implementations are to include both client and server functionality, as a solid unity.

The fact that CTP covers a number of layers, from transport layer to application layer, proves that the area of its responsibility starts from relatively low level and goes to a high one. So it shares a huge responsibility in network stack. Its share some responsibility of application layer in network stack. CTP/IP's relationship with the OSI-model [2] and UDP/IP ideology is shown on fig. 1.

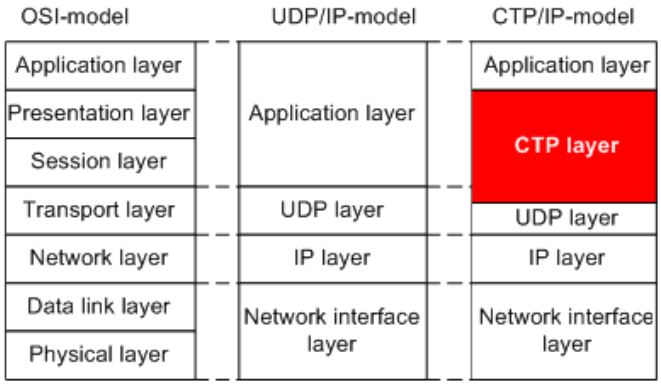

Figure 1: Relationship between OSI-model, UDP/IP-model and CTP/IP-model

3. CTP Implementation

All important transfer parameters, as IP addresses and ports of sender and recipient, are stored in the UDP header. Each packet can be fully identified by its sender, its receiver, and ID. Sender provides uniqueness of ID for each recipient in the following way: initial value of ID for next packet that will be sent is taken as pseudorandom number. After sending each packet, it is to be incremented. The very first packet sent to the recipient, is to be marked with special option to allow recipient to learn the value of the starting ID.

There are four storages for data, cumulated during lifetime, which provide functionality.

- *Session information storage*. It stores description of each workstation the current one communicates with. Among description, next packet ID, interchange timeout, and description of packets received from this recipient, are meant here.

Interchange timeout is used to determine when the sent packets need to be resend if they have not been confirmed. This timeout is adoptive (because a cluster can be rather heterogeneous and can involve workstations via both intranet and Internet). Initially, default timeout is taken (100 milliseconds, by default). After the first interchange, its value is taken as time, needed for it, multiplied by coefficient (3, by default).

If confirmation of packet's arrival will not be received during timeout, then the packet is to be resent. The period between resending will grow exponentially. If packet is not be confirmed after 8 re-sending (255 timeouts will pass), then an error message "Command is not confirmed too long" will be generated. If timeout is set to zero, then this feature is switched off.

Messages can be resent. So, it is necessary to protect the user from receiving one message several times. That is why; descriptions of received packets are stored for each addressee. It is implemented as an ordered list. First element contains the maximal ID of the packet, received in sequence. After this element, there can be more IDs, corresponding to packets, which have been received, but which are greater than the first element. After insertion of each new ID in this list, the sequence, which begins from the first element, is to be truncated. For example, let's assume that this storage contains {7, 9, 10, 11, 13, 14}. This means that all packets with ID less or equal to 7 and equal to 9, 10, 11, 13 and 14 already have been received. After receiving the packet with ID 8, the list will take the form {11, 13, 14}. If all packets arrive in sequence, then the list always contains a single element.

Values of IDs are to be in the endless loop (after 2^{32}-1 goes 0). Determining of starting ID, which was generated by the sender, is very important in this stage.

A new entry is added to session information storage when the first message is going to be sent or was received from workstation, or is unknown yet. There is to be a special entry for broadcasted messages.

- *Sent commands storage*. To send the command, packets are to be arranged. Some memory is to be allocated and

filled with packet headers and data. The fact is that it will not be freed and unallocated just after sending, but stored to the sent commands storage. A record can be removed from sent commands storage only after all its packets arrival has been confirmed. This ideology can be implemented not for "each command", but for "each packet" (like in CTP 1.0), but first variant is preferable. In this case, so named "smart buffers" can keep from redundant memory allocations, by reserving and guarding memory needed for headers, while doing packets data arrangement.

- *Large commands storage* is used to arrange the whole large message, when receiving it part by part. It stores the total amount, the vector of parts receiving status, and a buffer for compiling. Each part of the message, except, probably, the last one, is of maximal data size, so parts can easily find their places in the buffer, knowing their numbers. When all parts have been received, the message is considered to be arranged and the server informs the application about data arrival.
- *Deliveries storage*. The whole received message or error description is, so named, "delivery". After generating, they will be added into deliveries list. Then special deliverer threads will take them from the list and pass them to the application.

4. Parallel & Distributed Computing

- **Parallelism** is generally concerned with accomplishing a particular computation as fast as possible, exploiting multiple processors. **Parallel computing** is a type of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved at the same time. The scale of the processors may range from multiple arithmetical units inside a single processor, to multiple processors sharing memory, to distributing the computation on many computers. On the side of models of computation, parallelism is generally about using multiple simultaneous threads of computation internally, in order to compute a final result.
- *Parallelism* is also sometimes used for real-time reactive systems, which contain many processors that share a single master clock; such systems are fully deterministic.
- **Concurrency** is the study of computations with multiple threads of computation. Concurrency tends to come from the architecture of the software rather than from the architecture of the hardware. Software may be written to use concurrency in order to exploit hardware parallelism, but often the need is inherent in the software's behavior, to react to different asynchronous events (e.g. a computation thread that works independently of a user interface thread or a program that reacts to hardware interrupts by switching to an interrupt handler thread).
- **Distributed computing** studies separate processors connected by communication links. Whereas parallel processing models often (but not always) assume shared memory, distributed systems rely fundamentally on message passing. Distributed computing is a field of computer science that studies distributed systems. Distributed computing also refers to the use of distributed systems to solve computational problems. Distributed systems are inherently concurrent. Like concurrency, distribution is often part of the goal, not solely part of the solution if resources are in geographically distinct locations, the system is inherently distributed. Systems in which partial failures (of processor nodes or of communication links) are possible fall under this domain.

5. Conclusion

In this paper, we have proposed a self adaptive communication protocol for high performance peer to peer distributed computing. For rapid interchange between dozens of nodes are to be more pleasant for CTP ,because its activities will stay the same, but TCP will loose a lot on channels creating and recreating .For CTP, it does not matter who the recipient is. CTP's implementation doesn't use a critical amount of resources. Its overhead expenses are small enough to be ignored. This protocol has been implemented on a small network for the solution of nonlinear optimization problems, i.e. network flow problems. We plan to study a specification language for controller decision rules description. We shall also concentrate on the design of a decentralized environment for high performance peer to peer distributed computing. This type of Environment will permit one to use all the specificities offered by the peer to peer concept for high performance computing purpose. Self organization of peers for efficiency purpose or for insuring everlastingness of applications in hazardous situations or in the presence of faults will also be studied. The different applications considered will permit us to validate our protocol and decentralized environment in different high performance computing contexts.

References

[1] Gnutella Protocol Development. Hhttp://rfc-gnutella.sourceforge.netH.
[2] The FreeNet Network Projet. Hhttp://freenet.sourceforge.netH.
[3] D. El Baz, T. T. Nguyen et al, –CIP - Calcul intensif pair à pair", Poster, Colloque Ter@tec2009, SUPELEC, 30 Juin-1 Juillet 2009.
[4] N. Hutchison and L. L. Peterson, –The x-kernel: An architecture for implementing network protocols," in IEEE Transactions on Parallel and Distributed Systems, vol. 17, no. 1, pp. 64-76, 1991.
[5] H. Miranda, A. Pinto, and L. Rodrigues, –Appia: A flexible protocol kernel supporting multiple coordinated channels," in Proc. 21st International conference on Distributed Computing Systems (ICDCS- 21), (Phoenix, Arizona, USA), pp. 707–710, 2001.
[6] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu, –Coyote: a system for constructing fine-grain configurable communication services," in ACM Transactions on Computer Systems, 16(4): pp. 321–366, 1998.
[7] D. C. Schmidt, D. F. Box, and T. Suda, –ADAPTIVE—A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment," Journal of Concurrency: Practice and Experience, 5(4): pp. 269–

286, 1993.
[8] E. Exposito, P. Senac, M. Diaz, –FPTP: the XQoS aware and fully programmable transport protocol," in Networks, 2003. ICON2003. The 11th IEEE International Conference on, pp. 249-254.
[9] Matti A. Hiltunen, –The Cactus Approach to Building Configurable Middleware Services", in DSMGC2000, Nuremberg, Germany, 2000.
[10] G.T Wong, M.A Hiltunen, R.D Schlichting, –A configurable and extensible transport protocol," in Proceedings of IEEE INFOCOM '01, Anchorage, Alaska (2001), pp. 319–328.
[11] S. Floyd, T. Henderson, –The New-Reno Modification to TCP's Fast Recovery Algorithm," RFC 2582, Apr 1999.
[12] D. Leith and R. Shorten, –H-TCP protocol for high-speed long distance networks," in PFLDnet, Feb. 2004.
[13] D. El Baz, P. Spiteri, J. C. Miellou, D. Gazen, –Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems," in Journal of Parallel and Distributed Computing, Vol. 38, pp. 1-15, 1996.
[14] P. Owezarski, P. Berthou, Y. Labit, D. Gauchard, –LaasNetExp: a generic polymorphic platform for network emulation and experiments," in TridentCom'2008, Innsbruck, Austria, March 18-20, 2008.
[15] D. El Baz, G. Jourjon, –Some solutions for Peer to Peer Global Computing," in 13th Euromicro conference on Parallel, Distributed and Network-Base Processing, 2005, pp. 49-58

Author Profile

John pursuing M.Tech from BMU Rohtak (2014-2016) and received B.Tech degree in Computer Science and Technology from Maharshi Dayanand University in 2013. His main research interest includes: Social media, Artificial intelligence.