MassiveClicks: A Massively-Parallel Framework for Efficient Click Models Training

Skip Thijssen[1](✉), Pooya Khandel[1], Andrew Yates[1], and Ana-Lucia Varbanescu[1,2]

[1] University of Amsterdam, Amsterdam, The Netherlands
skip.thijssen@student.uva.nl
[2] University of Twente, Enschede, The Netherlands

Abstract. *Click logs* collect user interaction with information retrieval systems (e.g., search engines). Clicks therefore become implicit feedback for such systems, and are further used to train *click models*, which in turn improve the quality of search and recommendations results. Click models based on expectation maximization (EM) are known to be effective and robust against various biases.

Training EM-based models is challenging due to the size of click logs, and can take many hours when using sequential tools like PyClick. Alternatives, such as ParClick, employ parallelism and show significant speedup. However, ParClick only works on single-node multi-core systems. To further scale up and out, in this work we introduce MassiveClicks, the first massively parallel, distributed, multi-GPU framework for EM-based click-models training. MassiveClicks relies on efficient GPU kernels, balanced data-partitioning policies, and distributed computing to improve the performance of EM-based model training, outperforming ParClick by orders of magnitude when using GPUs and/or multiple nodes. Additionally, the framework supports heterogeneous GPU architectures, variable numbers of GPUs per node, allows for multi-node multi-core CPU-based training when no GPUs are available.

Keywords: massively-parallel training · multi-node multi-GPU · click models training · expectation-maximization models

1 Introduction

The primary goal of an Information Retrieval (IR) system, such as a search or recommendation engine, is to return results that satisfy users' information needs: users submit *queries* and receive *documents* in response to their query. Assessing the relevance of the documents (as answers to the query) is essential for the further development of IR systems. While domain experts can run such assessments, this approach does not scale with the number of IR applications and queries. Instead, Radlinski et al. [21] demonstrate that user interactions, such as *clicks*, may quantify users' satisfaction and serve as a proxy for relevance assessments. Thus, clicks are collected in *click logs* and further analysed.

D. Zeinalipour et al. (Eds.): Euro-Par 2023 Workshops, LNCS 14351, pp. 232–245, 2024.
https://doi.org/10.1007/978-3-031-50684-0_18

However, click logs are not perfect: user clicks suffer from various types of biases, such as position or popularity bias [7]. To mitigate biases, many click models have been developed, based on probabilistic graphical model (PGM) [7,18] and neural networks [3,19,24], and trained with users' click logs over time. In general, the PGM-based approaches are more explainable, more compact, and perform well, and are common for applications focusing on improving ranking models for web search [1,14,22,23].

While PGM-based click models are known to be effective, their training efficiency and scalability are not well studied, despite the immense volume of available search data - e.g., Google processes 1.2 trillion searches per year[1]. Scalability is essential to process increasingly large datasets, while efficiency is essential in light of raising concerns about the CO_2 footprint of big data applications [15]. Recently, Khandel et al. [16] introduced ParClick, a scalable algorithm for training Expectation-Maximization (EM)-based click models that enables multi-core training of PGM-based models, and demonstrates significant speed-up over previous methods. But ParClick's applicability is limited to multi-core shared-memory machines (basically, single-node CPU-only systems), and its performance is limited by the available number of cores.

In this work, we advance state-of-the-art with MassiveClicks , a framework that uses multi-scale, heterogeneous distributed systems for efficient EM-based training of click models. Our proposed framework supports the most common click models: Position-Based Model (PBM) [8], Click Chain Model (CCM) [13], User Browsing Model (UBM) [9], Dynamic Baysian Network Model (DBN) [6], and enables their training on heterogeneous system configurations, efficiently scales up to multiple nodes with multiple GPU devices, and, in the absence of GPU devices, can switch to CPU-based parallelism in a multi-node manner.

We measure the efficiency of MassiveClicks through extensive experiments using different mixes of CPUs and GPUs, and large-scale datasets. We find that MassiveClicks does use large-scale heterogeneous systems efficiently, and it can outperform ParClick by a factor larger than 850× when using 14 nodes with multiple GPUs.

This paper makes the following contributions:

- We design, implement, and analyze highly-efficient generic GPU kernels for training EM-based click models.
- We introduce MassiveClicks , the first framework that can be deployed to train click models using multi-node, multi-GPU, heterogeneous machine configurations.
- We demonstrate MassiveClicks provides orders-of-magnitude improvement over state-of-the-art in terms of training performance.
- We provide the open-source implementation of MassiveClicks.

[1] https://www.internetlivestats.com/google-search-statistics.

2 Background and Related Work

2.1 EM-Based Click Models

Click models are developed to address the bias in logged users' clicks over time. Formally, given a click model M, the probability of a click on the document d given the query q, will be calculated as follows [7]: $P(C_d) = P(E_d = 1) \cdot P(A_d = 1)$, where $P(E_d = 1)$ is the examination probability of document d, i.e., the probability that a user will scan this document on search engine results page (SERP) and $P(A_d = 1)$ is the attractiveness probability of d, i.e., the probability that user finds this document relevant for the input query. Various probabilistic graphical model (PGM)-based click models designed in the last decade differ in their assumption of user behavior while scanning the SERP, which in turn leads to specific formulations for the examination and attractiveness probabilities.

It follows that each model consists of a few examinations and a large number of attractiveness parameters. Their values can be estimated by employing the Expectation-Maximization (EM) algorithm through an iterative process where all parameters are initialized at the beginning, and new estimates are calculated based on user click logs at each iteration. This process continues for a certain number of iterations, such that the estimated parameters reach a convergence point for click probability calculation.

The primary challenge with EM-based click models training is that for every existing combination of query-document pairs, one attractiveness parameter exists; thereby, the number of attractiveness parameters scales with the size of the click log, and standard sequential training becomes too slow and inefficient. In this work, we propose to address this inefficiency by using all resources available in heterogeneous distributed systems.

2.2 Programming Models

In this work, we consider a heterogeneous node to combine a CPU and one or more GPUs. A distributed heterogeneous system is therefore a cluster of heterogeneous nodes, with potentially different architectures. For the remainder of this work, we use NVIDIA GPUs; however, our approach works without modification on, for example, AMD GPUs. The code, however, is not directly portable, but can be converted using, for example, the `hipify` tool[2].

To program distributed heterogeneous systems, we use a combination of CUDA (for GPUs), multi-threading (for CPUs), and MPI (for communication). Our framework is therefore usable on any multi-node multi-GPU cluster where CUDA and MPI are available.

2.3 Related Work

Click models were chiefly designed based on PGMs [7], and more recently, based on neural networks [3]. In the context of PGMs, Craswell et al. [8] introduced

[2] https://docs.amd.com/bundle/HIPify-Reference-Guide-v5.4/page/HIPify.html.

PBM that assumes the probability of clicking on a document in SERP highly relies on its position and is independent of the rest of the documents in the same SERP. In contrast, in CCM [13], the probability of clicking for a document is affected if the documents in higher rank within a SERP are clicked. Several other click models [7,12] are introduced as extensions of these models to improve their capabilities in addressing various biases in users' clicks, such as UBM [9] or DBN [6] click models. Our work is orthogonal to these approaches, as they only focus on the effectiveness of click models, while we focus on training efficiency and scalability aspects, but as they are trained with EM, our proposed framework is applicable for training them.

In contrast to click models' effectiveness, their training efficiency and scalability are not well studied. [20] focus on large-scale training of the Bayesian browsing model though it does not generalize with the rest of EM-based click models. Recently, [16] was introduced as a generic algorithm that can be applied for EM-based click models training; however, it is only limited to multi-core shared-memory machines. In our work, we aim to address the limitations of existing approaches, and we introduce the first general framework applicable to diverse machine configurations that can efficiently scale up when more (heterogeneous) resources are added.

Finally, there are several runtime systems - e.g., OmpSS, StarPU, IRIS, PARSEC or LAMA [2,4,5,11,17] - and many heterogeneous programming models [10]. However, our application is very specific in terms of computation to communication to synchronization ratios: the processing and code are very simple, and most design decisions are related to data structures, balanced data distribution, and synchronization. To the best of our knowledge, none of these systems could have better automated this design process. Therefore, we developed MassiveClicks as a prototype starting from the ParClick[3] method and code, and chose native programming models. However, for portability, one can easily port MassiveClicks to use different programming models or runtime systems.

3 Framework Design

Our goal is to design and prototype MassiveClicks as a general framework for training EM-based click models using distributed heterogeneous systems. The framework must support multi-GPU configurations (ranging from single-node/single-GPU to multi-node/multi-GPU), heterogeneous GPU architectures and memory sizes, and if/when a node does not have access to GPUs, the framework must run correctly on CPUs only. The framework must enable the users to partition the training dataset using several strategies, thus taking into account load balancing and processor compute capability. Finally, the framework must support four different click models: PBM, CCM, UBM, and DBN.

[3] https://github.com/uva-sne/ParClick.

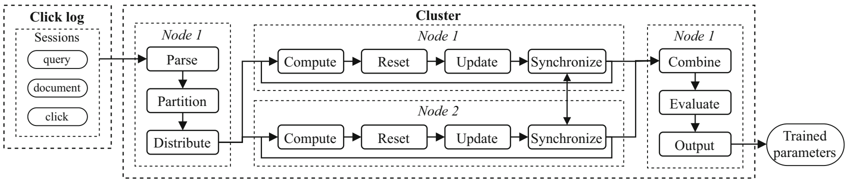

Fig. 1. The training process from start to finish.

A high-level design of MassiveClicks , complying with these requirements, is presented in Fig. 1. The training process is initiated by parsing the click log dataset, which includes queries, documents, and clicks on the documents, on a single machine. The dataset is then sorted internally and partitioned according to a user-specified partitioning scheme, with each partition being sent to a separate node in the cluster. A node receives the incoming partition and starts the training process by repeatedly carrying out the following four steps: (1) Estimating the click model parameters (*Compute*), (2) Clearing the previous iteration's results (*Reset*), (3) Copying the new iteration's results to the previous results (*Update*), and (4) Synchronizing the results with other nodes (*Synchronize*).

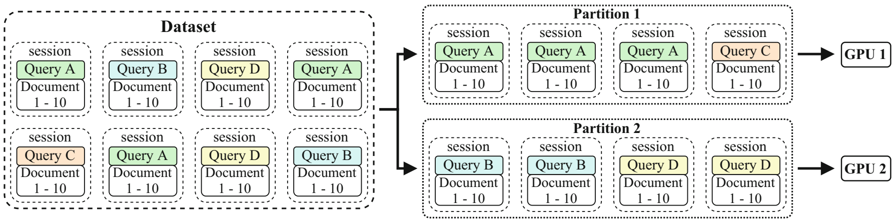

Fig. 2. Dataset distribution for separate GPUs, according to a partitioning scheme. Sessions are grouped by their search query into the same partition.

Upon completion of the designated number of training iterations, the node sends the trained parameters to the root node to evaluate the effectiveness (accuracy) of the trained click model.

4 Implementation Details

The implementation of the framework[4] utilizes multiple nodes, possibly with several GPUs per node, to train a click model using a click log dataset. The click log comprises search engine result pages (SERPs) - also referred to as sessions - and the clicks that occur on these pages. Each session is a personalized response

[4] MassiveClicks is open-source and available here https://github.com/skip-th/MassiveClicks.

generated by a search engine for a user's search query and contains a number of documents (search results) and the clicks on these documents.

Click models mainly use two types of parameters: (1) parameters common to documents with similar search queries from different sessions, and (2) parameters shared across all documents. The combination of these parameters allows for the estimation of the probability of a user clicking on a document given a certain query. Click models are trained precisely to estimate these parameters, where the estimation process requires calculations with different parameters on each document in the click log. Given the large number of documents, the application should be a well-suited workload for GPUs.

4.1 Data Distribution

We scale out MassiveClicks by using multiple GPUs (possibly on multiple nodes). To divide the click log among the multiple GPU devices, we propose four partitioning strategies, aiming to match different hardware configurations. These strategies aim to assign multiple sessions with the same search query to a single GPU device, as shown in Fig. 2, thereby eliminating the need for communication between nodes when estimating the parameters.

1. *Round-robin:* Assign groups of sessions to each device in a cyclic manner. It is suitable for test cases with smaller click logs, where slight imbalances in the workload per device do not significantly impact the performance.
2. *Maximum-Utilization:* Distribute, evenly, the number of sessions to all devices by continuously assigning groups of sessions to each GPU in turn. This strategy is useful for homogeneous (i.e., same-performance) GPUs.
3. *Proportional Maximum-Utilization:* Assign a number of sessions proportional to each device's memory size; devices with less memory receive fewer sessions, but have the same percentage of their memory occupied. This strategy is recommended when the size of a device's memory is an indicator of its performance.
4. *Newest Architecture First:* Prioritize assigning sessions to GPUs with the highest Compute Capability. Once these GPUs are "full", sessions are sent to less recent GPUs. This strategy is appropriate when there is a significant performance difference between older and newer GPUs in the system.

To minimize the memory footprint of the click log on the GPU, only the clicks on each document are included; the session, query, and document IDs are excluded during the transfer process.

4.2 GPU-Based Parameter Estimation

The estimation of the parameters for the click model is performed through iterative Expectation-Maximization (EM), which combines clicks on each session's documents and parameters from the previous iteration, stored in global GPU memory. An EM iteration has four phases: (1) *Computing* the parameters, (2) *Resetting* the original parameters, (3) *Updating* the original parameters, and (4) *Synchronizing* the result. The first iteration is initialized with a default value of $\frac{1}{2}$, as per the PyClick implementation[5].

Compute. The GPU processes one session per thread, with multiple threads handling sessions with the same search queries. Each thread handles all ten documents in a session sequentially. This granularity is a trade-off between the degree of parallelization and thread independence. A more fine-grained approach, with a single document per thread, would increase communication between threads, while assigning a group of sessions with a similar query to a single thread would decrease communication, but increase load imbalance, as groups may vary in size.

Each thread uses its ID to access the click log and retrieve the documents for its session. The parameters associated with a document are also stored in the click log and indexed for quick access. This index is precomputed on the host machine. The parameters are stored in the click log because different sessions processed by separate threads can contain documents belonging to the same search query and, thus, require the same parameters (such as attractiveness in the PBM click model). Estimating the parameters for these query-document (QD) pairs involves reading and writing to the same memory location, which is addressed by storing the parameter index alongside each document in the click log. Unfortunately, this approach does not coalesce memory accesses, but is the only currently viable solution.

The use of shared parameters by different GPU threads can result in race conditions, where one thread writes changes to a parameter before another thread has had a chance to read its original value. To prevent such errors, each thread writes its results to a unique intermediate parameter in global memory.

Update. The intermediate parameters are then combined into a single original parameter in the Update phase for use in the next iteration. This process starts by *resetting* the original values to the default value of $\frac{1}{2}$. A reset of the original parameters is necessary to ensure that the previous iteration's parameters do not influence the results of the next iteration twice. During the Update phase, there are two types of parameters being reduced: (1) the parameters that are unique to a group of similar-query sessions (such as attractiveness in PBM), and (2) the parameters that are shared by all threads (such as examination in PBM).

To update the unique intermediate parameters, each thread atomically writes the intermediate values to the corresponding original parameters, which are identified using the index stored in the click log. The intermediate shared parameters are also written atomically to the original parameters. However, writing all intermediate shared parameters simultaneously with all threads to the same small set of shared original parameters can cause a significant delay due to the large number of threads awaiting their turn to write. To reduce this delay, each thread

[5] https://github.com/markovi/PyClick/blob/master/pyclick/click_models/Param.py.

block performs a local reduction operation on each of its threads' intermediate shared parameters. The reduced results are then atomically written to the original parameters in global memory by a single thread. This writing thread also starts with the shared parameter whose index is equal to its thread block index, further reducing the number of atomic writes to the same memory location.

4.3 Communication and Synchronization

The *synchronization* of parameters between GPUs occurs once a GPU has finished updating all its parameters. For the shared parameters to be communicated between iterations and workers, they are transferred to the host machine. The aggregation of shared parameters happens first at node level, using only the parameters received the local GPUs. The node-level (combined) shared parameters are subsequently sent to the other nodes using an MPI `Allgather` routine. Each node then repeats the aggregation operation and copies the result to all its GPUs. A new iteration can then start.

4.4 Dealing with Heterogeneity

The framework integrates two programming models: CUDA and MPI, to perform the click model training. The two models are separated into distinct tasks, with CUDA handling the parallel computation on GPUs and MPI managing the communication and coordination between multiple nodes. The heterogeneity of combining CUDA and MPI is managed through C++ as the main programming language. In the framework, high performance is achieved by CUDA providing the parallel computing power required for training large click models, and MPI allowing for the distribution of work across multiple nodes, increasing scalability.

4.5 CPU-Based Parameter Estimation

Each click model is designed to run both on the CPU and the GPU, with instructions for host-side computation and device-side computation, respectively. The data partitioning strategies remain unchanged, with the data passed to each machine being the size of the corresponding (i.e., CPU or GPU) memory.

To train the click model, the nodes locally sort their assigned data partition into groups based on search query and distribute the groups evenly among the specified number of threads. This sorting helps to minimize communication between threads by reducing concurrent access to the same parameters. Further, the CPU-based approach assigns entire groups to a single thread (while the GPU-based approach assigns each session to its own thread to balance the load). Thus, the CPU-based approach can process the documents within a group of sessions sequentially, allowing for direct writes to the parameters unique to that group without the need for intermediate parameters as in the GPU-based approach. Parameters shared across all sessions still require atomic writes to retrieve the final result.

Each thread uses a local copy of the shared parameters to estimate their values; the original shared parameters are updated with the local copies when all threads have completed their work. Finally, the results are synchronized among multiple nodes using the same scheme as in the GPU-based training.

4.6 Limitations

The current implementation of MassiveClicks is limited by the size of the input machine's memory, which must contain the full click log. This is necessary because our data distribution requires that each GPU receives a set of sessions with "unique" queries - i.e., that do not occur on another GPU. This approach allows GPUs to compute the parameters for the session separately until the final synchronization step, when some parameters must be synchronized.

Currently, the input machine reads the entire click log, and groups same-query sessions together on the same GPU, according to the chosen partitioning scheme. The distribution of sessions to GPUs is performed only after reading the entire click log, so the size of each group of sessions is known to prevent the session groups from being assigned to GPUs with insufficient memory.

Assigning sessions to GPUs on separate nodes while reading the click log, in order to not be limited by the input machine's memory, requires the grouping of sessions located on different nodes. This operation is currently not supported.

The limitation can be circumvented by sorting the click log beforehand with sessions of the same query grouped together. The click log can then be processed in separate chunks that individually fit into the input machine's memory. The few parameters shared between all sessions can be combined afterward manually.

5 Evaluation

Our evaluation focuses on MassiveClicks's ability to use multi-node multi-GPU systems, its scalability, and its efficiency in using various heterogeneous configurations[6]. For all our experiments, *training time* is the main metric of interest, and we also report speed-up versus state-of-the-art (to quantify advancements in performance) and memory footprint (to assess feasibility for large datasets).

Table 1. Training time, speed-up, and ACE for **PBM** and **CCM** for different datasets and **NVIDIA RTX A4000 GPUs** compared to ParClick on an **AMD EPYC 7402P CPU** with **48 threads**.

GPUs	D10						D25						D50					
	Training [s]		Speed-up		ACE [%]		Training [s]		Speed-up		ACE [%]		Training [s]		Speed-up		ACE [%]	
	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM	PBM	CCM
1	4.0	6.1	10.1	73.1	14.7	42.4	10.1	15.7	10.5	70.1	14.0	41.4	21.8	31.9	9.7	66.3	13.5	40.5
2	1.9	3.3	20.9	137.6	10.4	41.7	5.8	8.5	19.2	129.5	12.9	39.8	12.1	17.4	17.5	121.8	12.8	39.0
4	1.1	1.6	37.5	272.2	13.7	39.6	2.8	4.1	39.4	267.5	13.1	39.2	6.1	7.6	34.5	280.0	7.1	24.5
8	0.6	0.9	67.1	488.8	12.4	36.2	1.7	2.2	66.4	494.1	10.1	36.2	3.0	4.4	71.2	482.0	12.9	36.3
14	0.4	0.5	110.1	850.4	12.2	37.2	1.0	1.4	110.0	794.0	10.8	33.4	1.9	2.7	112.1	790.7	11.6	34.6

[6] Additional evaluation data and plots are available at https://bit.ly/HP23-extra.

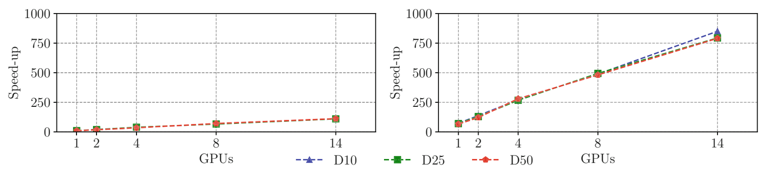

Fig. 3. MassiveClicks's speedup over ParClick for PBM (left) and CCM (right) using 10, 25, and 50 million sessions. ParClick runs on an AMD EPYC 7402P CPU with 48 threads, and MassiveClicks uses NVIDIA RTX A4000 GPUs.

5.1 Experimental Setup

The Platform. For all our experiments, we use DAS-6[7], a cluster comprising compute nodes with one CPU (AMD EPYC 7402P or Intel E5-2630 v3) and various NVIDIA GPUs (e.g., Titan X, Titan-X Pascal, A6000, A4000), interconnected using 100 Gbit/s Ethernet and managed by SLURM. The nodes run CUDA v11.5.119 on the Red Hat 8.5.0-3 operating system. We measure GPU (kernel) performance using NVIDIA Nsight Compute NVIDIA[8], and we measure the CPU performance with simple code instrumentation.

The Data. All our models are trained and tested with the Yandex dataset[9]. For all experiments, we divide the dataset into two parts: the first 80% forms a training set, and the last 20% forms a test set. Because the trained click-model only provides relevant parameters for search results contained within the training set [7, p.53], we filter the test set to only include sessions from the training set. To do so, we follow ParClick's approach of *filtering on sessions* [16].

5.2 The GPU Impact

Speed-Up. To assess the performance gain due to GPU acceleration, we analyze the speed-up of MassiveClicks on up to 14 GPUs compared to ParClick running on a 24-core AMD EPYC 7402P CPU. We use PBM and CCM as representatives of a less and more complex click model, respectively. Speed-up is calculated as the ratio of ParClick's training time to MassiveClicks's training time. We also measure the **A**verage Percentage of **C**omputation Time spent within the parameter estimation stage per it**E**ration (ACE - introduced in [16]), to indicate parameter synchronization overhead.

[7] https://www.cs.vu.nl/das/.

[8] https://developer.nvidia.com/nsight-compute.

[9] https://www.kaggle.com/competitions/yandex-personalized-web-search-challenge/data.

Table 1 shows the training time, speed-up, and ACE, for 10, 25, and 50 million sessions. We observe that a single GPU already improves training performance up to 10x for PBM and 73x for CCM. When using 14 GPUs, MassiveClicks shows speed-ups as high as 850x for the smallest dataset, and as high as 790x for the largest log. However, the stages unrelated to parameter estimation, measured with ACE, take up a considerable part of the total training time. Consequently, the maximum speed-up in training time observed in Table 1 differs significantly between click models, as also seen in Fig. 3: PBM achieves a significantly lower maximum speed-up compared to CCM, due to the lower computation intensity of the former. As expected, higher computational intensity indicates more efficient GPU acceleration, and thus higher speed-up.

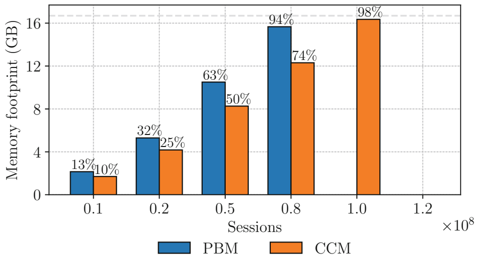

Fig. 4. MassiveClicks memory footprint for **PBM** and **CCM** using an **NVIDIA RTX A4000 GPU** with 16.7 GBs of memory and up to 120M sessions.

Memory Footprint. Figure 4 shows that the memory footprint linearly increases with the number of sessions, albeit with different slopes. The PBM and CCM click models cut off at 75 and 100 million sessions, respectively, when the GPU's memory becomes too small to hold the entire click log and parameters.

5.3 Scalability

We measure the scalability of MassiveClicks by comparing speed-up and training time PBM and CCM for an increasing number of NVIDIA RTX A4000 GPUs computing up to 120 million sessions, the maximum size of the Yandex dataset.

Figure 5 shows that even for very large click logs, our framework significantly reduces training time. The addition of multiple GPUs further reduces training time to several seconds for even the largest dataset sizes. These performance improvements through scaling the number of GPUs also apply to the less complex PBM click model. Furthermore, the speed-up shows that the overhead of scaling to multi-GPU multi-node configurations has minimal impact on the final training time. Each click log size scales close to the ideal scaling factor for both the complex CCM and less complex PBM click model.

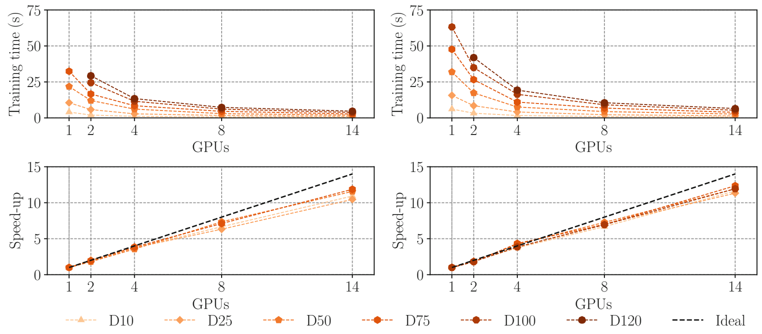

Fig. 5. MassiveClicks training time (top) and scalability (bottom) for **PBM** (left) and **CCM** (right) using 14 **NVIDIA RTX A4000 GPUs** with up to 120M sessions.

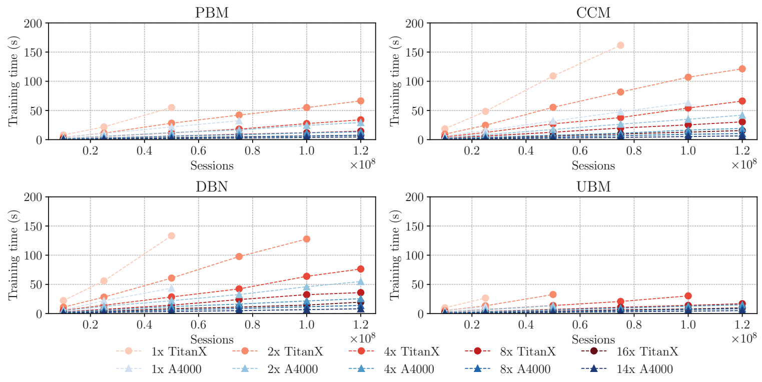

Fig. 6. MassiveClicks training time for **PBM**, **CCM**, **DBN**, and **UBM** on up to 16 NVIDIA TITAN X (red) and 14 A4000 (blue) GPUs for dataset sizes of D10 to D120. (Color figure online)

5.4 Putting It All Together

We demonstrate the framework's full capability by testing various combinations of click models computed on multiple nodes with different GPU devices; the achieved performance is summarized in Fig. 6. The results indicate a significant difference in training time when comparing the older NVIDIA TITAN X GPU to the newer A4000 model: A4000 performs the training significantly faster. However, the TITAN X does still provide significant performance improvements and is capable of reducing training time, even for the largest dataset, to seconds.

These results show that multi-GPU training of EM-based click models with our proposed framework can significantly reduce training time. Moreover, the performance benefits of GPUs are more significant for models like the CCM and DBN, which are more complex models than PBM and UBM.

6 Conclusion

User clicks can be used as implicit feedback to IR systems such as search and recommendation engines, but they suffer from various types of biases. Click models are trained to avoid such biases, but they can be slow and inefficient., with the notable exception of ParClick [16], the first parallel, shared-memory solution for training these models. As ParClick is fundamentally limited in scale, we propose MassiveClicks , a framework for training EM-based click models on heterogeneous systems of any scale. MassiveClicks supports heterogeneous, multi-node and multi-GPU systems, but can be also used with CPU-only clusters. We also support multiple partitioning schemes.

Our empirical analysis demonstrates that MassiveClicks can speed-up training by more than 850× when compared with ParClick. The framework is open-source and can be executed on any distributed heterogeneous system using NVIDIA GPUs, where CUDA and MPI are available.

In the future, we plan to improve the heterogeneity of training by employing CPU and GPU resources simultaneously, a task that is particularly challenging in determining the best partitioning of user clicks among processors. We aim to also extend the framework by implementing more efficient data ingestion and data distribution mechanisms.

References

1. Agarwal, A., et al.: Estimating position bias without intrusive interventions. In: WSDM'19, pp. 474–482 (2019)
2. Augonnet, C., et al.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. In: CCPE Special Issue: Euro-Par 2009 23, 2 February 2011, pp. 187–198. https://doi.org/10.1002/cpe.1631
3. Borisov, A., et al.: A neural click model for web search. In: WWW '16, pp. 531–541 (2016)
4. Bosilca, G., et al.: PaRSEC: exploiting heterogeneity to enhance scalability. Comput. Sci. Eng. **15**(6), 36–45 (2013). https://doi.org/10.1109/MCSE.2013.98
5. Bueno, J., et al.: Productive programming of gpu clusters with OmpSs. In: IEEE IPDPS'12, pp. 557–568 (2012). https://doi.org/10.1109/IPDPS.2012.58
6. Chapelle, O., Zhang, Y.: A dynamic Bayesian network click model for web search ranking. In: ACM WWW '09, pp. 1–10. Association for Computing Machinery (2009)
7. Chuklin, A., et al.: Click Models for Web Search. Morgan & Claypool, San Rafael (2015)
8. Craswell, N., et al.: An experimental comparison of click position-bias models. In: ACM WSDM '08, pp. 87–94 (2008)
9. Dupret, G.E., Piwowarski, B.: A user browsing model to predict search engine click data from past observations. In: ACM SIGIR '08, pp. 331–338 (2008)
10. Fang, J., et al.: Parallel programming models for heterogeneous many-cores: a comprehensive survey. CCF Trans. High Perform. Comput. **2**(4), 382–400 (2020). ISSN 2524-4930. https://doi.org/10.1007/s42514-020-00039-4

11. Fraunhofer SCAI: LAMA - development of fast and scalable software. https://www.scai.fraunhofer.de/en/business-research-areas/high-performance-computing/products/lama.html
12. Grotov, A., Chuklin, A., Markov, I., Stout, L., Xumara, F., de Rijke, M.: A comparative study of click models for web search. In: Mothe, J., et al. (eds.) CLEF 2015. LNCS, vol. 9283, pp. 78–90. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24027-5_7
13. Guo, F., et al.: Click chain model in web search. In: ACM WWW'09, pp. 11–20 (2009)
14. Joachims, T., et al.: Unbiased learning-to-rank with biased feedback. In: ACM WSDM '17, pp. 781–789 (2017)
15. Jones, N., et al.: How to stop data centres from gobbling up the world's electricity. Nature **561**(7722), 163–166 (2018)
16. Khandel, P., et al.: ParClick: a scalable algorithm for EM-based click models. In: ACM Web 2022.WWW'22, pp. 392–400. Virtual Event, Lyon, France: ACM (2022). ISBN 9781450390965. https://doi.org/10.1145/3485447.3511967
17. Kim, J., et al.: IRIS: a portable runtime system exploiting multiple heterogeneous programming systems. In: IEEE HPEC'21, pp. 1–8 (2021). https://doi.org/10.1109/HPEC49654.2021.9622873
18. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning. The MIT Press, Cambridge (2009)
19. Lin, J., et al.: A graph-enhanced click model for web search. In: ACM SIGIR '21, pp. 1259–1268. New York, NY, USA (2021)
20. Liu, C., et al.: BBM: Bayesian browsing model from petabyte-scale data. In: ACM KDD'09, pp. 537–546. Paris, France (2009)
21. Radlinski, F., et al.: How does clickthrough data reflect retrieval quality? In: ACM CIKM'08, pp. 43–52. New York, NY, USA: ACM (2008)
22. Vardasbi, A., et al.: Cascade model-based propensity estimation for counterfactual learning to rank. In: ACM SIGIR'20, pp. 2089–2092. Association for Computing Machinery (2020)
23. Wang, X., et al.: Position bias estimation for unbiased learning to rank in personal search. In: ACM WSDM'18, pp. 610–618 (2018)
24. Yu, H.-T., et al.: A rank-biased neural network model for click modeling. In: CHIIR '19 (2019)