The Network Data Handling War: MySQL vs. NfDump

Rick Hofstede, Anna Sperotto, Tiago Fioreze, and Aiko Pras

University of Twente
Centre for Telematics and Information Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Design and Analysis of Communications Systems (DACS)
Enschede, The Netherlands
r.j.hofstede@student.utwente.nl, {a.sperotto,t.fioreze,a.pras}@utwente.nl

Abstract. Network monitoring plays a crucial role in any network management environment. Especially nowadays, with network speed and load constantly increasing, more and more data needs to be collected and efficiently processed. In highly interactive network monitoring systems, a quick response time from information sources turns out to be a crucial requirement. However, for data sets in the order of several GBs, this goal becomes difficult to achieve. In this paper, we present our operational experience in dealing with large amounts of network data. In particular, we focus on MySQL and NfDump, testing their capabilities under different usage scenarios and increasing data set sizes.

1 Introduction

Computer networks are growing in size and complexity, resulting in a network load that is constantly increasing [1, 2]. In such a scenario, network monitoring can provide vital information about the health of the network's communication infrastructure. Such information can then be used to achieve a satisfactory network operability. Therefore, network monitoring is a crucial activity in many network management solutions.

Several techniques for monitoring network traffic are available today, each one having a particular purpose and highlighting different aspects of network traffic information. Some of them focus on collecting information about individual packets, such as Tcpdump [3], while others focus on information about flows (*i.e.*, metadata information about sets of packets), such as NetFlow [4].

Independent of the involved level of abstraction, most of these techniques rely on storage points in order to collect and analyze network data. On this subject, diverse solutions are available, such as databases [5, 6], single-file data storage [3, 7] or multiple-file data storage [8]. In this paper, we report about our operational experience with MySQL [5] and NfDump [8] while dealing with data sets of several GBs. MySQL has the advantage to be a well-known Database Management System (DBMS) and to offer the full potentiality of the SQL language. It is extensively used by Web applications [9], but it has also been employed as

F.A. Aagesen and S.J. Knapskog (Eds.): EUNICE 2010, LNCS 6164, pp. 167–176, 2010.

an information source for network traffic monitoring [10]. On the other hand, NfDump specifically targets the problem of network data storage and processing. It indeed stores network data into binary files without the context of a DBMS. NfDump is well known in the network community and commonly employed to collect network information [11] [12].

Both information sources (MySQL and NfDump) deal relatively well with small amounts of data, but few is known when they have to handle larger amounts. We therefore want to give an answer to the following research question: *What are the differences in performance between MySQL and NfDump when handling large data sets?* In order to give an answer, we measured the response time of the two systems on increasingly large data sets. For this comparison, we used 24 hours of network traffic from the University of Twente (UT) [13] network, which was converted to both information sources' storage formats.

The remainder of this paper is structured as follows. In Section 2 we review the current state of the art on works considering different network information sources. Section 3 describes the data set used for our information source analysis. In Section 4 we describe the used methodology by presenting our measurements. Section 5 presents the results of our comparison. Finally, we close this paper in Section 6, where we draw our conclusions.

2 Related Work

In literature, several studies on the performance of information sources for network monitoring have been proposed. Siekkinen *et al.* [14] presents a DBMS-based solution, called InTraBase, which provides the infrastructure for analysis and management of data and metadata obtained from network measurements. The authors compare the processing times of the InTraBase approach to a file-based approach, namely Tcptrace [7]. They conclude that for relatively small source files, Tcptrace is the best choice. However, for improved manageability and scalability, a DBMS approach based on PostgreSQL [6] is advocated. Similarly to them, our investigation is also based on a DBMS approach, MySQL. However, we concentrate on a multi-file approach, NfDump. It is also worth mentioning that our considered network data set is three times larger (around 30 GB) than the data set considered in Siekkinen's work (10 GB).

Similarly to us, Kobayashi *et al.* [15] presents a comparison between a file-based approach for storing network traffic information and a DBMS approach. NfDump was used as a representative of the file-based approach, while MySQL and PostgreSQL were the tested implementations of the DBMS approach. The comparison was divided into two categories: 1) storing time, and 2) search and display time. Regarding storing time, the authors concluded that a file-based approach was in general faster than the DBMS approach. For the search and display times, the file-based approach was relatively slow compared to the DBMS. While the research in [15] is concentrated on short term data storage, we are interested in analyzing performance of information sources when handling historical data. In addition, we will focus our measurements only on search time.

Besides handling large amounts of network data by using an ordinary DBMS or a file-based approach, also decentralized setups can be used to speed up the query process. One approach is described in [16], where decentralized nodes perform data aggregation, which leads to shorter query execution times. However, in our work, we do not take distributed solutions into consideration.

3 Data Collection Process

This section describes how we captured, processed and stored the network traffic used in our experiments. After that, we provide statistics about this data set.

The University of Twente (UT) main router exports network information about flows[1] in the form of NetFlow v9 records. The NetFlow records were collected using a *nfcapd* process, part of the NfDump suite [8]. It creates binary files of 5 minutes of network data, in the order in which it is captured. After that, the data was post-processed and imported into a MySQL database, also ordered by time. NfDump offers a routine for reading and displaying *nfcapd* binary files.

The MySQL schema used for storing network data follows the description of a flow. Each flow is stored as one flow record, ordered by its start time and having the following attributes: *flowid*, *start_time*, *end_time*, *duration*, *ipv4_src*, *port_src*, *ipv4_dst*, *port_dst*, *protocol*, *tos*, *packets* and *octets*. Besides these attributes, a MySQL table has a standard index on a table's primary key, which is *flowid* in our case. Moreover, our table also contains some other indexes to improve the speed of operations on this table. The available indexes are as follows: *start_time*, *tuple* (*ipv4_src*, *port_src*, *ipv4_dst*, *port_dst*, *protocol*), *packets*, *octets* and *duration*.

The data set used in our measurements consists of exactly 24 hours of network data, collected on September 18th, 2008, between 0:00 and 23:55. It consists of roughly 445.000.000 flow records. The disk space needed for MySQL data is 31 GB, while its indexes need around 43GB of disk space. On the other hand, the NfDump data needs about 22 GB of disk space.

4 Methodology

In this section we present the methodology used to measure query response times on our data set. We define query response time as the time between invoking a query on a data set and having retrieved the complete result set. However, we do not print the result set to the screen. All measurements have been performed on a machine with an Intel Dual Core Xeon 3070 CPU, 4GB DDR2 RAM and 2 SATA disks in RAID0 (striping). The first subsection will describe various approaches used in our data analysis, while the second subsection defines several case studies considered in our comparison.

[1] We consider a flow as "a set of packets passing by an observation point in a network during a certain time interval and having a set of common properties" [17].

4.1 Approaches to Data Analysis

MySQL and NfDump were compared by measuring their query response times when handling large amounts of network data. While from NfDump's side no optimization can be set up, MySQL provides the use of indexes, which results in more efficient access of data. In some cases indexes can considerably increase a database table's performance. The indexes are generally transparent to the end user, but it is possible to force or forbid the use of any of these. If an index does not exist or is not used, MySQL will perform a sequential scan over the whole table. However, if the index exists, MySQL can quickly determine the position to seek for in the middle of the data file without having to look at all the data.

Since indexes can affect the performance of a query, we tested MySQL both with and without indexes. This leads us to a total of four considered approaches:

1. *MySQL*: MySQL decides whether or not to use an available index.
2. *MySQL with indexes*: MySQL is forced to use a specified index.
3. *MySQL without indexes*: MySQL is forced to ignore any index.
4. *NfDump*: NfDump behaves as it is designed for.

4.2 Case Studies

The four approaches aforementioned were observed while triggering different types of operations on the considered network data. These operations, from now on addressed as case studies, are as follows:

1. *Listing*: This case study causes the information sources to do a sequential scan over the data set, without doing any calculation or filtering. Actually, there is a filtering action on the *start_time* attribute, but since we assume this as our basic situation, we consider it as part of the listing.
2. *Listing + Filtering*: The result of this case study only contains the flow records of the first case study that have destination port 22. We chose to filter on this port because this traffic is fairly distributed over the day and thus also over the entire data set. The amount of traffic with destination port 80 for instance, would depend too much on the time of the day.
3. *Grouping + Filtering 1*: This case study groups all flow records of the "Listing + Filtering" case study by their source IP addresses and calculates the sum of octets for each group.
4. *Grouping + Filtering 2*: Instead of grouping by just one attribute, this query groups by five attributes and calculates the sum of octets for each group. We want to verify here whether grouping by multiple attributes will require significantly more time than the previous case study.

In order to find out how MySQL and NfDump behave depending on the size of our data set, we tested their performance on time-incremental basis (with data slices of one hour of network data). In this case, incremental means that the first test is executed on a data set from 12 AM to 1 AM, the second on a data set from 12 AM to 2 AM, and so on. The data set is thus becoming larger with

Table 1. Queries representing our case studies

Case study	MySQL query	NfDump query
Listing	SELECT * FROM table WHERE start_time BETWEEN x AND y	nfdump -M data_dir -T -R nfcapd.date1:nfcapd.date2 -o long
Listing + Filtering	SELECT * FROM table WHERE start_time BETWEEN x AND y AND port_dst = 22	nfdump -M data_dir -T -R nfcapd.date1:nfcapd.date2 -o long "dst port 22"
Grouping + Filtering 1	SELECT ipv4_src, SUM(octets) FROM table WHERE start_time BETWEEN x AND y AND port_dst = 22 GROUP BY ipv4_src	nfdump -M data_dir -T -R nfcapd.date1:nfcapd.date2 -o long -a -A srcip "dst port 22"
Grouping + Filtering 2	SELECT ipv4_src, port_src, ipv4_dst, port_dst, protocol, SUM(octets) FROM table WHERE start_time BETWEEN x AND y AND port_dst = 22 GROUP BY ipv4_src, port_src, ipv4_dst, port_dst, protocol	nfdump -M data_dir -T -R nfcapd.date1:nfcapd.date2 -o long -a -A srcip,srcport, dstip,dstport,proto "dst port 22"

every query execution, until the full data set is used. This results in a total of 24 tests per approach, for each case study.

Table 1 shows the syntax of the queries representing the case studies. All SQL queries have a *start_time BETWEEN x and y* statement in their *WHERE* clause. This statement is needed to select the current time-incremental data set to be tested. The value of x is constant in all our tests. The same can be done with NfDump using the *nfcapd.date1:nfcapd.date2* statement, where the input files of the data set have to be specified. It is important to notice here that it is not possible to filter on a flow's *start_time* or *end_time* using NfDump's query syntax. The only way to do this, is by selecting another input file set.

In the particular case of MySQL, we need to know which index MySQL is going to use for a specific query. MySQL's "EXPLAIN" command shows which indexes *could* be used and which indexes are *actually going to* be used. We used this information to force or forbid indexes in our case studies. Finally, note that all query response times were measured without considering screen printing.

Table 1 only reports the query for the *MySQL* approach and the used command for the *NfDump* approach. For the remaining approaches (*MySQL with indexes* and *MySQL without indexes*) the query is the same plus the use of the "FORCE INDEX" and "IGNORE INDEX" keywords.

5 Results

This section presents the results of our measurements. They show the response times of each approach for each case study. The plots describe the relation between the size of the data set (on the horizontal axis) and the query response

time in minutes (on the vertical axis). All tests were executed three times, to reduce the effect of disturbances (*e.g.*, unforeseen processes running on our test machine). The plots are based on the averages of each pair of three query executions. Moreover, error bars were added representing the standard error value:

$$e = \frac{\sqrt{\frac{\sum (X_i - \bar{X})^2}{(n-1)}}}{\sqrt{n}}$$

5.1 Listing and Listing + Filtering

Figure 1 shows the results of the "Listing" case study. A first observation, related to the *MySQL without indexes* approach, is that the result is not constant. Since MySQL would have to do a sequential scan over the whole data set, which would take the same amount of time for each query iteration, this is contradictory to what we expected. Instead, there seems to be an almost linear relationship between the size of the data set (hours of network data) and the query response time, namely between 7 and 21 hours of data.

We can thus conclude that there is an overhead in writing the result set, even though we took care to avoid screen output to reduce the impact of such operations on the response times. Before and after the period of linearity, the data density will be less. Otherwise, the whole trace would be linear. This can be explained due to the fact that during night (*i.e.*, from 9 PM to 7 AM) there is much less traffic transiting within the UT network.

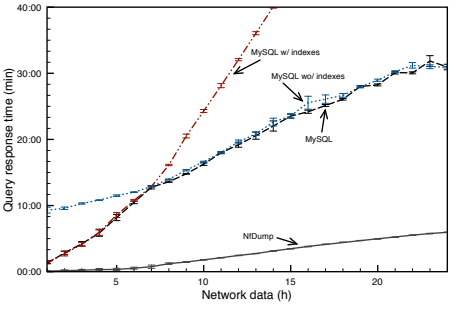

Fig. 1. Listing queries

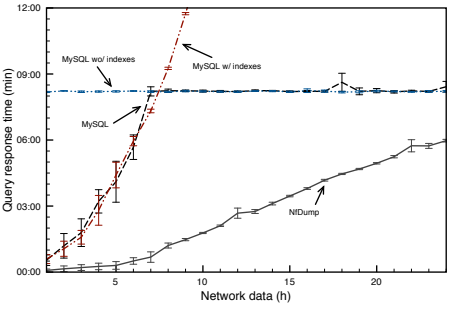

Fig. 2. Listing + Filtering queries

A second observation is that after using a data set of approximately 7 hours of network data, *MySQL with indexes* needs more execution time than *MySQL without indexes*. The reason for that comes from the fact that MySQL has to read too much data from the disk (*i.e.*, from the index and the data), which takes more time than doing a full sequential scan over the data set. Moreover, *MySQL* makes the right choice about when to use the index: exactly from the point where the query response time of *MySQL without indexes* is less than with a forced use of the index, *MySQL* chooses to discard the index.

The queries that were executed using NfDump had the shortest query response times during all queries of the "Listing" case study. The reason for this is that NfDump uses small input files, which are concatenated as specified in the queries. In contrast to this, MySQL has to process the whole data set during every query execution.

On its turn, Figure 2 shows the results of the "Listing + Filtering" case study. Using *MySQL without indexes*, the result is now indeed a (constant) horizontal line. Since the query response times are much less than in the "Listing" case study, we can conclude that MySQL is first filtering on *dst_port = 22*, before a selection based on the start times of the flows is made. That result set is much smaller (231.098 flow records) than the average result set of the "Listing" case study (18.445.740 flow records).

After about 7 hours of network data, *MySQL* decides not to use indexes anymore. This is, according to the query response times, the correct decision. We can also notice that until that point, the standard error value is much larger than after it. As Schwartz *et al.* describe in [18], the MySQL query cache is completely in-memory, which is managed by MySQL itself. This means that the response time of the second and third test until 7 hours of data is shorter because the results are already in memory. After 7 hours of data, the response times stabilized because MySQL cannot take advantage of its cache anymore.

Once more, NfDump has the shortest query response times. Its behavior is closely related to MySQL's behavior in the "Listing" case study: from 7 AM until 9 PM there is a linear relationship between the size of the used data set and the query response time. Before and after that period, the data density is relatively less. Therefore, NfDump's query response times are not linear to the size of the used data set over the whole trace.

5.2 Grouping + Filtering 1 and Grouping + Filtering 2

Figures 3 and 4 show the results of the two "Grouping + Filtering" case studies.

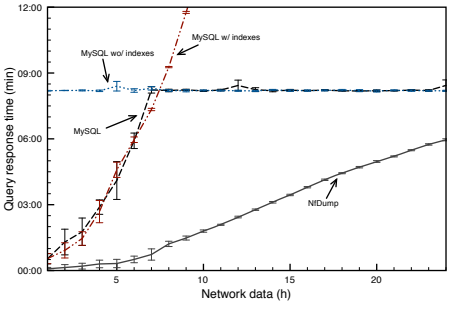

Fig. 3. Grouping + Filtering 1 queries

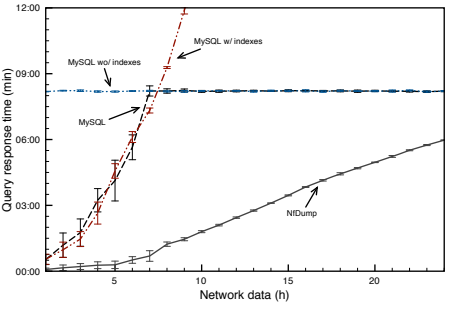

Fig. 4. Grouping + Filtering 2 queries

As previously mentioned, we expected the "Grouping + Filtering 2" case study in general to take longer to complete than "Grouping + Filtering 1".

However, our results show that this is not the case: both case studies take exactly the same execution time. Moreover, these times are also identical to the response times of the "Listing + Filtering" case study.

The "Grouping + Filtering" case studies require the data set to be sorted by all attributes specified in the *GROUP BY* clause. Our results suggest that once the data is retrieved, grouping by five attributes is not more costly than grouping by one. Since the response times for both case studies are identical, we can conclude that data retrieval and the disk operations related to it are the bottleneck of the data set manipulations. Likewise previous case studies, NfDump outperforms MySQL once more.

5.3 Double-Sized Data Set

All figures presented before suggest that there might be an intersection between MySQL's and NfDump's query response times, somewhere between 24 and 48 hours of network data. To get more insight into this speculation, we tested all case studies on a 48 hour data set. The results of the "Listing + Filtering" case study can be found in Figure 5. We do only discuss the "Listing + Filtering" case study here, since the results of the others show the same behavior compared to their 24 hour counterparts. Additionally, Figure 6 was created using the response times on 24 and 48 hours of network data of both *MySQL wo/ indexes* and *NfDump*. The two lines were created using linear regression. As shown in Figure 6, MySQL's and NfDump's query response times will never intersect, since the response times are diverging from each other.

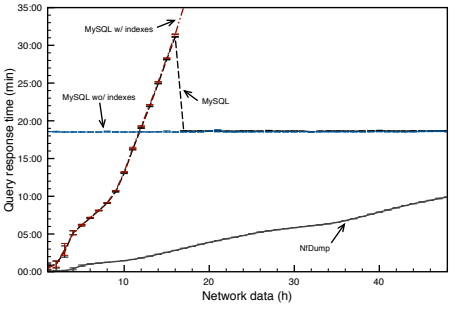

Fig. 5. Listing + Filtering [48h data set] queries

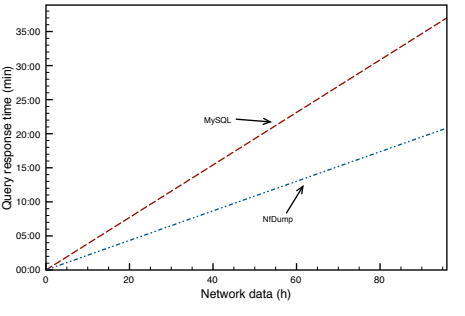

Fig. 6. Relationship between MySQL's and NfDump's response times

By comparing the result of the 24 hour data set with its 48 hour counterpart, we can make some interesting observations. First, the behavior of *MySQL wo/ indexes* is the same, but its query response times are doubled. On the contrary, *NfDump*'s query response times are not doubled (*i.e.*, 6 minutes compared to 10 minutes, on a 24 hour and a 48 hour data set, respectively).

A second observation is that after 12 hours of network data, *MySQL* is not able to correctly decide to ignore the available index after that point. Note that on a 24 hour network data set, *MySQL* was able to make the correct decision. Remarkable is that the point where *MySQL* decides to discard the index (around 16 hours of network data), is around twice that point on a 24 hour data set.

More importantly, it can be observed that there is still no intersection between MySQL's and NfDump's query response times. For a complete data set, the response time of *MySQL wo/ indexes* is the shortest for our case studies. If we compare the 48 hour data set to the original 24 hour one, we observe that:

- *MySQL wo/ indexes*' response time grows by a constant factor each time the data set is enlarged. In particular, the response time is doubled here.
- *NfDump*'s query response time grows linearly according to the amount of data processed, but it is not influenced by the size of the complete data set. Moreover, the response time on a 48 hour data set is less than doubled, compared to the response time on a 24 hour data set.

Considering the implementations of MySQL and NfDump, we expect the relationship between the two approaches to be maintained also for larger data sets. Since the query response times are diverging from each other, we assume that MySQL's and NfDump's query response times will most likely never intersect.

6 Conclusions

This paper presented a comparison between the performance of MySQL and NfDump when handling data sets consisting of several GBs of network information. We measured the response time of both systems on a data set of 24 hours of flow data in an incremental manner and keeping into account several usage scenarios. Moreover, since indexes can improve MySQL's performance, we also considered them while defining the usage cases.

Siekkinen *et al.* [14] advocated the use of a DBMS as the best solution for network information for the sake of data management. Differently, our measurement results indicate NfDump as being the best solution to query large network data sets when observing response time. In all our comparisons NfDump outperformed MySQL. Therefore, we advise, when the response time is a striking issue, to make use of NfDump or similarly designed multiple-file based approaches.

Despite NfDump outperformed MySQL in all our comparisons, we observed that MySQL's response time was constant after considering 7 hours of network data, whereas NfDump's response time increased. In order to verify if NfDump's response times would intersect MySQL's on a larger data set, we enlarged our data set to 48 hours of data. Even then, NfDump outperformed MySQL in all tests. However, while MySQL's response times were doubled, NfDump's were slightly close to double. Taking this behavior as a pattern to even larger data sets, we can assume that NfDump's response times will most likely be shorter than MySQL's.

Acknowledgements

This research work has been supported by the EC IST-EMANICS Network of Excellence (#26854). Special thanks to Djoerd Hiemstra and Ramin Sadre for their valuable contribution to the research process.

References

1. Steinder, M., Sethi, A.S.: A survey of fault localization techniques in computer networks. Science of Computer Programming 53(2), 165–194 (2004)
2. Casey, E.: Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. Digital Investigation 1(1), 28–43 (2004)
3. Tcpdump/libpcap (April 2010), http://www.tcpdump.org/
4. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, Informational (2004)
5. MySQL (April 2010), http://www.mysql.com/
6. PostgreSQL (April 2010), http://www.postgresql.org/
7. Tcptrace (April 2010), http://www.tcptrace.org/
8. NfDump (April 2010), http://nfdump.sourceforge.net/
9. Liu, X., Heo, J., Sha, L.: Modeling 3-Tiered Web Applications. In: Proc. of the 13th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 307–310 (2005)
10. Hofstede, R., Fioreze, T.: SURFmap: A Network Monitoring Tool Based on the Google Maps API. In: Application session proc. of the 11th IFIP/IEEE Int. Symp. on Integrated Network Management, pp. 676–690. IEEE Computer Society Press, Los Alamitos (2009)
11. Li, Y., Slagell, A., Luo, K., Yurcik, W.: CANINE: A combined conversion and anonymization tool for processing NetFlows for security. In: Proc. of 10th Int. Conf. on Telecommunication Systems, Modeling and Analysis (2005)
12. Minarik, P., Dymacek, T.: NetFlow Data Visualization Based on Graphs. In: Goodall, J.R., Conti, G., Ma, K.-L. (eds.) VizSec 2008. LNCS, vol. 5210, pp. 144–151. Springer, Heidelberg (2008)
13. University of Twente (April 2010), http://www.utwente.nl
14. Siekkinen, M., Biersack, E.W., Urvoy-Keller, G., Goebel, V., Plagemann, T.: InTraBase: Integrated traffic analysis based on a database management system. In: Proc. of the End-to-End Monitoring Techniques and Services, Washington, DC, USA, pp. 32–46. IEEE Computer Society, Los Alamitos (2005)
15. Kobayashi, A., Matsubara, D., Kimura, S., Saitou, M., Hirokawa, Y., Sakamoto, H., Ishibashi, K., Yamamoto, K.: A Proposal of Large-Scale Traffic Monitoring System Using Flow Concentrators. In: Kim, Y.-T., Takano, M. (eds.) APNOMS 2006. LNCS, vol. 4238, pp. 53–62. Springer, Heidelberg (2006)
16. Lim, K.S., Stadler, R.: Real-time views of network traffic using decentralized management. In: Proc. of the 9th IFIP/IEEE Int. Symp. on Integrated Network Management, Nice, France, pp. 119–132 (2005)
17. Quittek, J., Zseby, T., Claise, B., Zander, S.: Requirements for IP Flow Information Export (IPFIX). RFC 3917, Informational (2004)
18. Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, J., Lentz, A., Balling, D.J.: High performance MySQL, 2nd edn. O'Reilly, Sebastopol (2008)